commodore

The Transactor

Commodore Canada's
Tech/News Periodical

VOLUME 3
Issue #5

Bits & Pieces

ON GOTO ELSE

A useful sequence in BASIC is IF - THEN - ELSE. Unfortunately, as everyone knows, PET BASIC can't do an IF THEN ELSE, or can it? Well, the answer is no, it can't. However, a lot of times the 'THEN' keyword is followed by a line number which really means GOTO. IF GOTO ELSE we can do!

The common approach in BASIC is an IF statement followed by some 'if criteria', followed by THEN or GOTO and a line number. Anything beyond here will be ignored since if the condition is satisfied, the GOTO is executed, and if not, BASIC drops down to the next line of the program. Therefore, the next line usually contains the 'ELSE' code. For example:

```
100 IF X=B GOTO 120 : X=X+1 : GOTO 100
```

...will never work. Even though it does absolutely nothing, to do it correctly we need:

```
100 IF X=B GOTO 120
110 X=X+1 : GOTO 100
```

To get it all on one line we use the ON GOTO statement:

```
100 ON -(X=B) GOTO 120 : X=X+1 : GOTO 100
```

True/false logic in PET BASIC produces a "0" for false and "-1" for true (try PRINT 4=5, 5=5). A negative argument will give ?ILLEGAL QUANTITY ERROR so we change the sign with -(A=B). Fortunately, an argument that is out of range (ie.'0')for ON GOTO will not cause execution to drop to the next line but rather continue with the next statement. This is also true if there are not enough line numbers following the GOTO to satisfy the argument. For example:

```
100 ON PEEK(32768+X) GOTO 120 : X=X+1 :GOTO 100
```

...will GOTO 120 only if the character PEEKed is an "A".

Index Transactor #5

Bits & Pieces 1
ON GOTO ELSE 1
MisguideINT 3
PET/CBM EPROMs 3
Invade Invaders 4
80/40 Vision 4
Backup 9
BASIC Plotter 10
Machine Language Monitor Intro 12
The 6845 Video Controller 18
Getting Usable Video Signals 20
PRINT-AT Routine 21
The Print Mint 22
Programming Tips 24

VIC-20 Bonus Section

VIC-20 Cartridge Development 29
A Little VIC Music 32
* Another Voice For The VIC-20 34
* Joystick Control on The VIC-20 36
* Computer Magic 38
VIC Loader For PET/CBM 41
SUPERMON For The VIC 49

VIC-20 Memory Maps 58

* Reprinted from
Commodore, The Microcomputer Magazine

When you think about it, an IF THEN ELSE is only useful if you can fit the IF criteria, the THEN criteria and the ELSE criteria all on one line, which is often not possible. Using this variation of ON GOTO, you'll get the IF and ELSE criteria on one line; the THEN conditions will go elsewhere. Additionally, you can have several THEN conditions just by adding more line numbers.

MisguideINT

In algebra, the integer of a decimal number is defined as the next lower whole number past the fractional part. Therefore:

```
PRINT INT (1.3)
 1
PRINT INT (-1.3)
-2
```

One might expect the second example to return "-1". But BASIC is doing it right. If you want the next lower whole in order of magnitude, you'll have to code:

```
PRINT INT (ABS (-1.3)) * SGN(-1.3)
```

Of course "-1.3" will probably be a variable in your program. One last note... integer variables automatically do an INT operation on decimal numbers:

```
A% = INT (A)    is the same as
A% = A
```

PET/CBM EPROMs

EPROMs come in all shapes and sizes but not all work with PET/CBMs. The most useful ones are the 4K and 2K sizes. There is also a 1K EPROM which you could probably get quite cheap, but this is getting somewhat small and wastefull of valuable address space.

The part numbers are set up so that the last 2 digits represent the number of K bits, so this number divided by 8 gives K bytes. Here is a list of EPROMs that DO work with PET/CBMs:

2532 - 4K byte
2716 - 2K byte
2516 - same chip as above
2708 - 1K byte
2508 - same chip as above

Several manufacturers (Texas Intruments, Motorola, Hitachi, etc.) produce these chips so you might find some letters before or after the part number. The 2516s and 2716s are virtually the same IC (ie pin for pin compatible), but the 2732s and 2532s are quite different. The 2732 won't operate in the PET due to power supply requirements. The T.I. TMS2716 is also incompatible.

Any local electronics shop should have availability information and EPROM programmers with software are available through some Commodore dealers or see the ads in Compute!, BYTE, etc.

Invade Invaders

Paul Higginbottom has a quick note for 'Invaders' players (from Midnight Software Gazette):

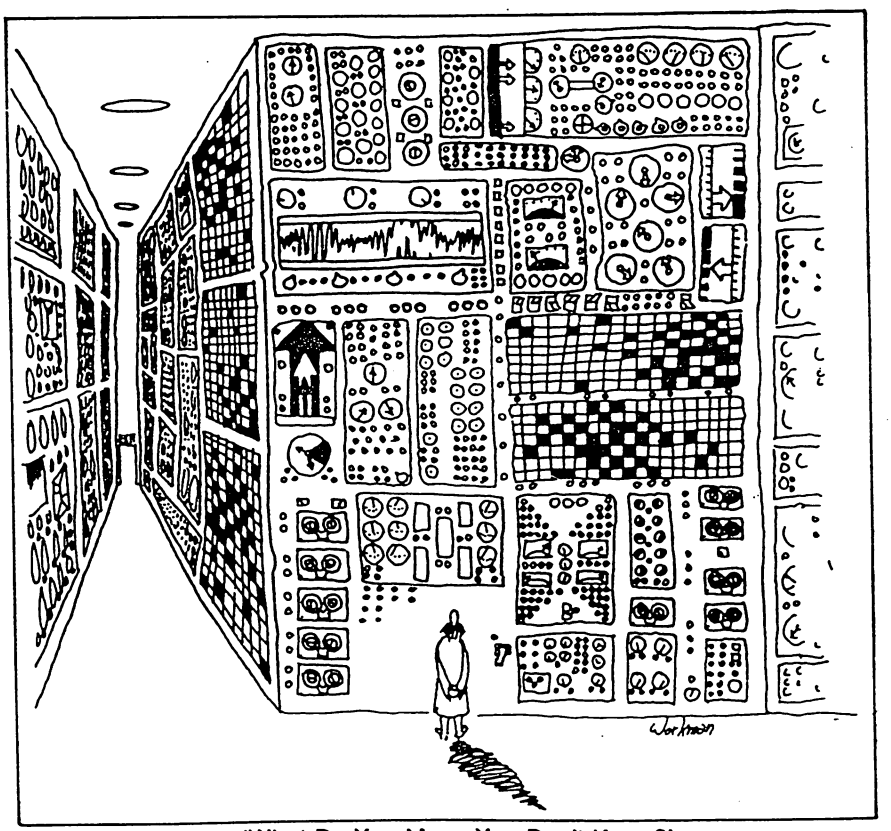
```
$0E01 (3585) - sets speed you move & fire; normally 2
$0E09 (3593) - sets invader firing speed ; normally 4
$0E0E (3598) - sets mother ship speed    ; normally 6
```

Finally, $0623 (1571) contains the character after missiles to erase them. Try POKEing with 102 ($66) for Wall Invaders.

80/40 Vision

Jim Butterfield doesn't have bionic eyes but he does have a way to test for 80 or 40 column screen:

```
WD=80 : POKE 32768+1024, 96 : IF PEEK(32767)=96 THEN WD=40
```



'What Do You Mean You Don't Know?'

BackPack

BATTERY BACKUP SYSTEM

FOR COMMODORE PET/CBM COMPUTERS

Standard Features:

- Full power to PET/CBM for a minimum of 15 minutes
- Installs within PET/CBM cabinet
- No wiring changes necessary
- Batteries recharged from PET/CBM integral power supply

Specifications:

- **Physical Size:** 5.5" x 3.6" x 2.4"
- **Weight:** 4.5 lbs.
- **Time to reach full charge:** 16 hours
- **Duration of outputs:** Minimum of 15 min.
- **Voltages:** +16, +9, −12, −9
- **Battery Life Expectancy:** 3 to 5 years
- **Battery On-Off Switch**

For Use With:

- Commodore PET/CBM 2001 and 4000 series computer
- Commodore PET/CBM 8000 series computer (screen size will not be normal on battery back-up)
- Commodore C2N Cassette Drive

Never again lose valuable data because of power shortages or line surges. **BackPack** supplies a minimum of 15 minutes reserve power to 32K of memory, the video screen and tape drive. **BackPack** fits inside the PET/CBM cabinet and can be installed easily by even the novice user. **BackPack** is recharged during normal operation and has an integral on-off switch.

BackPack comes fully assembled and tested. Instructions included.

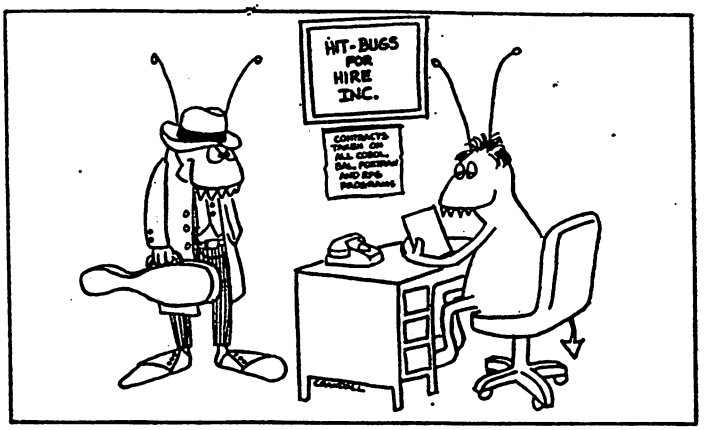
BackPack is a trademark of ETC Corporation
CBM/PET are trademarks of Commodore Business Machines

Designed and manufactured by:

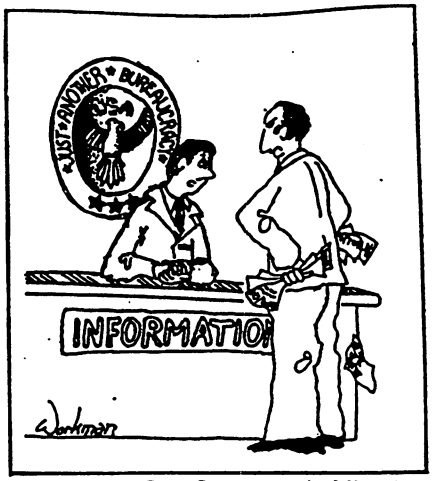
ELECTRONIC TECHNOLOGY CORPORATION
P.O. Box G, Old N.C. 42
Apex, North Carolina 27502
Phone: (919)362-4200 or (919)362-5671

Electronic Manufacturing
Technical Design and Development
Computer System Technology

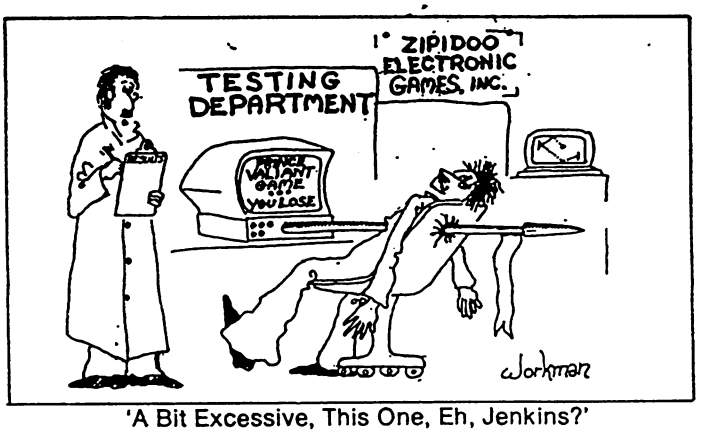
ETC ELECTRONIC TECHNOLOGY CORPORATION



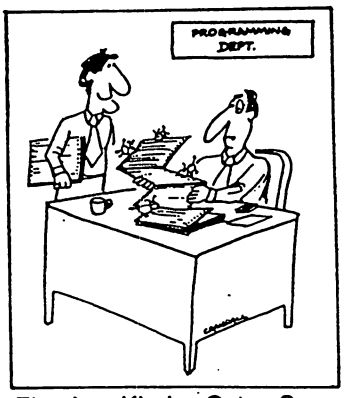
'The Spats and Violin Case Won't Be Necessary in This Line of Work, Perkins.'



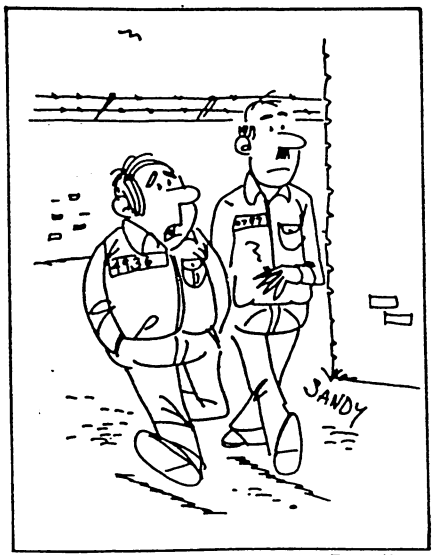
'I Know It's Our Computer's Mistake, Mr. Hill, But It Would Be Easier in the Long Run if You Did Change Your Name to ZP4/QE/70K.'



'A Bit Excessive, This One, Eh, Jenkins?'

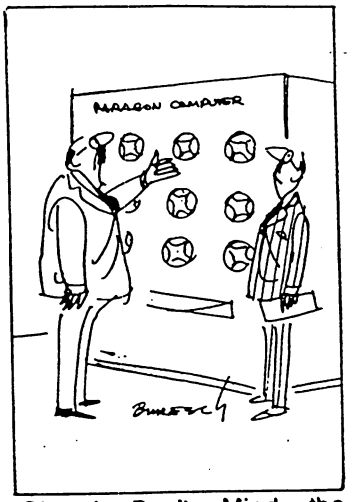


'They're Kinda Cute Once You Get Used to Them.'

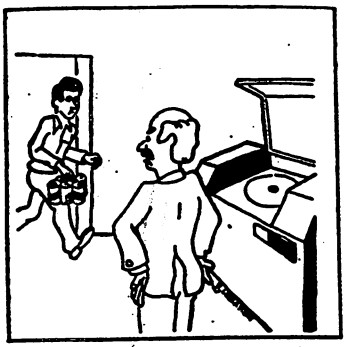


'Murder, eh? I'm in for Bending, Folding and Mutilating Computer Cards.'

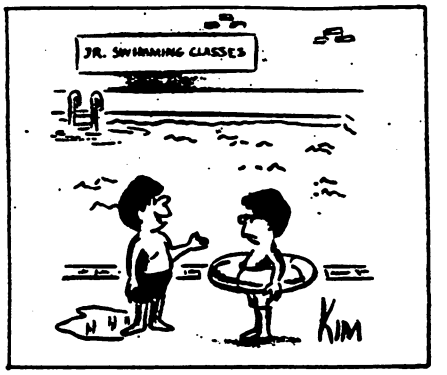
'Our Computer Is on the Blink. Can You Send Over a Hundred of Your Fastest Mathematicians?'



'Oh, I Don't Mind the $800,000 Price. But Is That the Only Color It Comes In?'

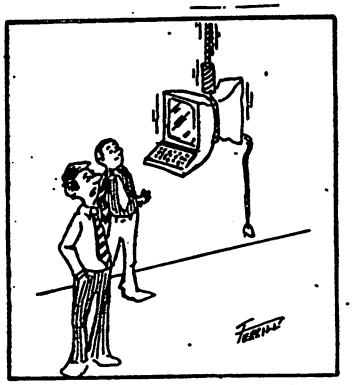


'I Said I Needed Another DISK Pack.'

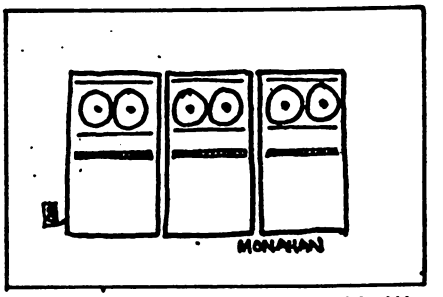


'There's Nothing to It. Just Remember That Force Equals Mass Times Acceleration.'

'I Think a Simple Error Message Would Suffice.'

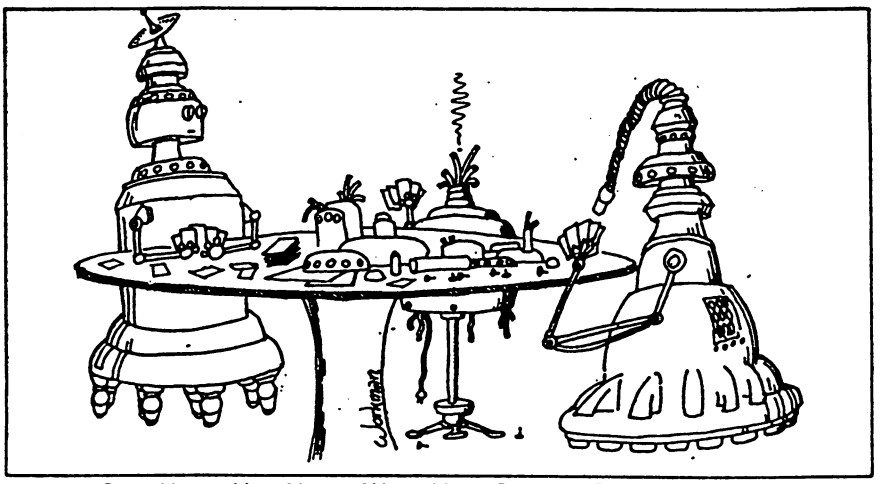


'Obviously, the Work of an Irate User.'

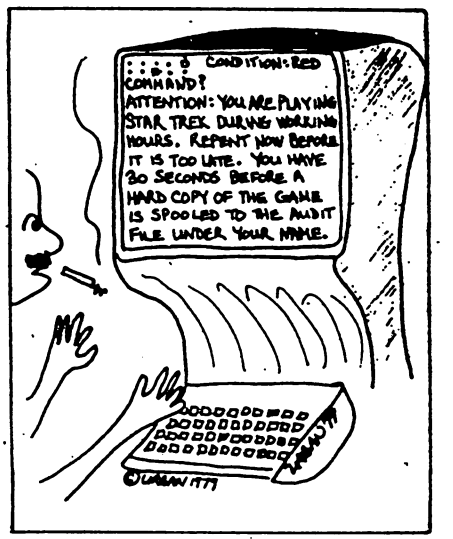


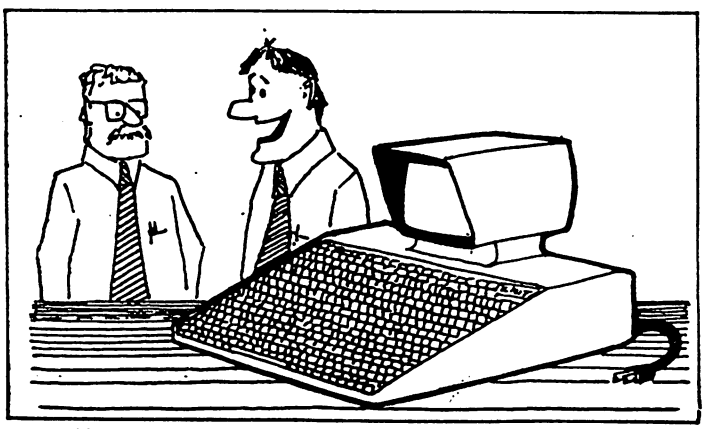
'Then It's Agreed. At 12:35 P.M., We All Break Down Just for Fun.'

'How About Kicking Off With a Good, Old-Fashioned Memory Dump?'

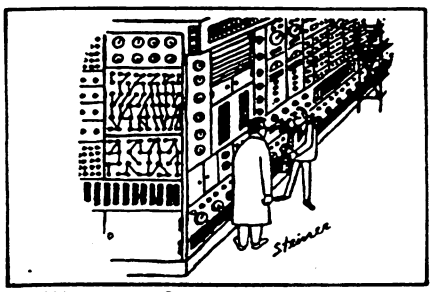


'Gee, Nano, You Never Were Very Good at Strip Poker.'





'Here You See Our Entry to the Far East Market.'



'It Wants a Squirt of Oil on That Squeaky Door Hinge Back There.'

Floppy BackPack

BATTERY BACKUP SYSTEM

FOR COMMODORE DUAL DRIVE FLOPPY DISK

Standard Features:

- Full power to Commodore Dual Drive Floppy Disk for a minimum of 15 minutes
- Installs within Disk Drive cabinet
- No wiring changes necessary
- Batteries recharged from Disk Drive's internal power supply

Specifications:

- Physical Size: 5.5'' x 3.6'' x 2.4''
- Weight: 4.5 lbs.
- Time to Reach Full Charge: 16 hours
- Duration of Outputs: Minimum of 15 min.
- Voltages: +16, +8
- Battery Life Expectancy: 3 to 5 years
- Battery On-Off Switch

For Use With:

- Commodore CBM 2040 Dual Disk Drive
- Commodore CBM 4040 Dual Disk Drive
- Commodore CBM 8050 Dual Disk Drive

Floppy BackPack is a total, rechargeable, battery backup system for the Commodore line of Dual Drive Floppy Disks. Used in conjunction with BackPack (battery backup for the PET/CBM) it is now possible to save valuable data to either disk drive during power shortages and line surges. **Floppy BackPack** also reduces chances of disk damage in the 2040.

Floppy BackPack fits inside the disk cabinet and can be installed easily by even the novice user. **Floppy BackPack** is recharged during normal operation from the disk's own internal power supply and has its own integral on-off switch.

Floppy BackPack comes fully assembled and tested. Instructions included.

Floppy BackPack is a trademark of ETC Corporation
CBM is a trademark of Commodore Business Machines

Designed and manufactured by:

ELECTRONIC TECHNOLOGY CORPORATION
P.O. Box G, Old N.C. 42
Apex, North Carolina 27502
Phone: (919)362-4200 or (919)362-5671

Electronic Manufacturing
Technical Design and Development
Computer System Technology

ELECTRONIC TECHNOLOGY CORPORATION

Backup

Apologies to Dave Hook for omitting his Romswitch & Utility Switch BASIC Loader for the SWARM-100 Board. Two issues later, here it is.

```
0 REM    ROMSWITCH & UTILITY SWITCH
1 REM          FOR SWARM-100
2 REM
3 REM(C) DAVID A. HOOK, 58 STEEL STREET
4 REM     BARRIE, ONTARIO, CANADA
5 REM    L4M 2E9    (705) 726-8126
6 REM
7 REM       ALL RIGHTS RESERVED
8 REM PERMISSION TO COPY FOR NON-COMMERCIAL PURPOSES
9 REM       AS OF JUNE 14, 1981
10 SA=897:B=64
11 PRINT"[CLR RVS]ROMSWITCH / UTILSWITCH   FOR  SWARM-100
12 PRINT"[DN DN]DO YOU WANT TO LOAD AT [RVS]";SA"[OFF ']Y[CL]";
13 POKE167,0
14 GETZ$:IFZ$=""THEN14
15 PRINTZ$:IFZ$<>"N"THEN29
16 PRINT"[DN]PUT IN HIGH MEMORY [RVS]Y[CL]";
17 POKE167,0
18 GETZ$:IFZ$=""THEN18
19 PRINTZ$:IFZ$="N"THEN23
20 EA=PEEK(52)+256*PEEK(53)-2:SA=EA-B:J%=SA/256:J=SA-256*J%
21 POKE 52,J:POKE 53,J%:POKE 48,J:POKE 49,J%
22 GOTO29
23 INPUT"[DN]START ADDRESS  ?[CL CL CL]";Z$
24 IFZ$="?"THENPRINT"[UP UP UP]":GOTO23
25 SA=VAL(Z$)
26 IFASC(Z$)<>36THEN29
27 Z$=RIGHT$("000"+MID$(Z$,2),4)
28 FORI=1TO4:Z=ASC(MID$(Z$,I,1)):Z=Z+7*(Z>57)-48:SA=SA+Z*16^(4-I):NEXT
29 EA=SA+B:TA=SA+42:TA%=TA/256:RL=SA+30
30 FOR A=SA TO EA:READ D:POKE A,D:NEXT
31 POKE RL+1,TA%:POKE RL,TA-TA%*256
32 PRINT"[CLR DN]NOTE THE 'SYS' ROUTINE ADDRESSES:
33 PRINT"[DN DN]FOR: SOFT ROM SWAP -- SYS("SA")
34 PRINT"[DN]      HARD ROM SWAP -- SYS("SA+4")
35 PRINT"[DN]      UTILITY SWAP  -- SYS("SA+56")
36 DATA 162,   5, 208,   2, 162,   0
37 DATA 160,   4, 165, 144, 201,  85        "[CLR]"   =  clear screen
38 DATA 208,   2, 160,   0, 120, 153        "[HOME]"  =  cursor home
39 DATA  56, 232, 138, 208,   3, 108        "[UP]"    =  cursor up
40 DATA 252, 255, 152,  10, 168, 185        "[DN]"    =  cursor down
41 DATA   0,   0, 149, 144, 200, 202        "[CL]"    =  cursor left
42 DATA  16, 247,  88, 108, 250, 255        "[CR]"    =  cursor right
43 DATA 195, 137, 253,  23, 230,  46        "[RVS]"   =  reverse mode on
44 DATA   0,   0, 179, 255, 212, 120        "[OFF]"   =  reverse mode off
45 DATA 228,  85, 120, 141,  60, 232        "[ ' ]"   =  1 space
46 DATA 141,  56, 232,  88,  96             "[15DN]"  =  15 cursor downs
```

BASIC Plotter

Paul Higginbottom
Commodore Canada

This program will plot random lines using the "quarter-square" graphics characters. Although it's a program in itself, it could easily be made into a subroutine.

The program has been set up for 80 column screens (line 9040). Notice "LL" (Line Length) is multiplied by 2 in lines 2020 & 2030? Since the quarter squares use up half a character space in the "x" direction, an 80 column screen can have up to 160 "half-characters" horizontally. Similarly, on 25 lines there can be up to 50 half characters vertically ("y" direction). For 40 column screens you'll need to change LL to 40; the second parameter remains the same since both have 25 lines.

Line 2000 clears the window (if one set), the screen, and sets graphics mode (no gap between lines). If you like, substitute CHR$(142) with 'esc-rvs-N' and stick it inside the quotes.

```
2000 PRINT"[HM HM CLR]"CHR$(142)
2010 GOSUB 9000
2020 X1=INT(RND(TI)*LL*2) : Y1=INT(RND(TI)*50)
2030 X2=INT(RND(TI)*LL*2) : Y2=INT(RND(TI)*50)
2040 GOSUB 3000 : Y1=Y2 : X1=X2 : GOTO 2030
3000 REM **********  PLOT A LINE  **********
3010 DX=X2-X1 : DY=Y2-Y1 : X=X1 : Y=Y1
3020 L=SQR(DX*DX+DY*DY) : IF L=0 THEN 3040
3030 XI=DX/L : YI=DY/L
3040 GOSUB 8000 : IF (ABS(X2-X)<=ABS(XI)) AND
     (ABS(Y2-Y)<=ABS(YI)) THEN RETURN
3050 X=X+XI : Y=Y+YI : GOTO 3040
8000 REM **********   PLOT X, Y   **********
8010 TX=INT(X+IR):TY=INT(Y+IR) :SQ=AM(TX AND AM, TY AND AM)
8020 P=BS+TX/DV-INT(TY/DV)*LL : POKE P, C(I(PEEK(P))OR SQ)
     : RETURN
9000 REM **********     SETUP     **********
9010 DIM C(15), I(255), AM(1,1)
9020 FOR I=0 TO 15 : READ C(I) : I(C(I))=I : NEXT
9030 FOR I=0TO1 : FOR J=0TO1 : AM(J,I)=(J+1)*4↑I : NEXTJ,I
9040 LL=80 : BS=32768+24*LL : DV=2 : AM=1 : IR=.5
9050 DATA  32, 123, 108,  98, 126,  97, 127, 252,
          124, 255, 225, 254, 226, 236, 251, 160
9060 RETURN
```

The subroutine at 9000 sets up an array with the 16 possible combinations of the quarter squares. BS is the base address or the POKE address of the bottom left corner of the screen.

All plotting efforts are performed by the two subroutines at 3000 & 8000. Subroutine 3000 plots a line from x1,y1 to x2,y2 by plotting several points (sub 8000). At the same time, subroutine 8000 must determine if there is already a point in a character space. If there is, the POKE information must not interfere with existing points.

Lines 200X are used for plot criteria generation. The above merely plots random lines. For something more meaningful, try substituting with these:

```
2020 X1=0 : Y1=1
2025 FOR X2=0 TO 159
2030 Y2=EXP (X2/31.4)
2040 GOSUB 3000 : Y1=Y2 : X1=X2 : NEXT : END

2020 N=6 : C=3.1415926/160 : X1=0 : Y1=25
2025 FOR X2=0 TO 159
2030 Y2=25 + 24 * SIN(X2 * N * C)
2040 GOSUB 3000 : Y1=Y2 : X1=X2 : NEXT : END

2020 N=8 : C=3.1415926/160 : X1=0 : Y1=50 : DC=100
2025 FOR X2=0 TO 159
2030 Y2=25 + 24 * COS(X2 * N * C) * EXP(-X/DC)
2040 GOSUB 3000 : Y1=Y2 : X1=X2 : NEXT : END
```

The first plots an exponential curve. Notice the Y origin is set to 1 rather than 0. This accounts for a slight inaccuracy as the plotter draws horizontal lines using the top "half-character" rather than the bottom half-character. This could be changed by modifying the character table at 9050.

The second draws a SINE curve starting half way up the screen (Y1=25). The variable N represents the number of half cycles displayed (N=6 will draw 3 complete cycles).

The last one is a decaying COSINE wave, origin at top-left (Y1=50). For higher decay rates, use lower values in DC.

Finally, with little effort you could use the plotter routine to draw axes for your functions.

'Now, You Say the Last Time You Saw It Was in a Macro You Were Coding...'

The Machine Language Monitor - An Introduction

When the first 2001 Series PET hit the market back in 1978, the first programs Commodore released were Squiggle, Bigtime and The Machine Language Monitor. Back then, the M.L.M. was loaded from tape. Commodore decided to introduce the M.L.M. to the next version of BASIC. The two became great companions and have been rommies ever since! (ugh. So much for PET folklore!)

The Machine Language Monitor is quite simply a utility program for performing more direct operations on machine memory and processor registers. For example, the BASIC SAVE command is used to store the contents of BASIC text space. The M.L.M. 'S' command can be used to save ANY part of memory because the start and end addresses are given by the user, not pre-determined by the machine.

In this article, I'll attempt to give a brief overview of the main sections of the M.L.M, plus some explanation of the 6502 microprocessor registers. For more details on machine language, there are several good books on the subject. Three are: MOS 6502 Programming Manual (see your dealer), 6502 Assembly Language Programming by Lance A. Leventhal (any good computer store), and the PET Machine Language Guide by Arnie Lee (Abacus Software).

Engaging the M.L.M.

There are two ways to enter the M.L.M. program. They are known as "break to the monitor" and "call to the monitor". In 6502 machine language, a BRK (break) instruction is represented by hex 00. Therefore, a SYS to any location in memory that contains a zero, will result in a "break to the monitor":

```
SYS 1024    ;brackets around
SYS 4        address unnecessary
```

The contents of decimal locations 1024 and 4 are always zero (unless deliberately altered). For example:

```
SYS4

B*
     PC  IRQ  SR AC XR YR SP
.;  0005 E455 30 00 5E 04 F8
.
```

Notice the 'B*' at the left margin. This indicates break to the monitor. To "call" the monitor, one must SYS to the location in ROM at which the monitor program starts:

```
SYS 64785  :BASIC 2.0
SYS 54386  :BASIC 4.0
```

A 'C*' will be displayed for call to the monitor.

M.L.M. Commands

There are 6 M.L.M. commands:

R - display Registers
M - display Memory
G - Go or execute memory
X - Exit to BASIC
L - Load
S - Save

Several monitor extension programs have been released that offer extra commands in addition to the above six. Programs like NEWMON, SUPERMON and EXTRAMON are loaded into RAM and link themselves in with the M.L.M. in ROM to produce a more sophisticated monitor utility. However, for now we'll concentrate on the resident commands.

R - Display Registers

When the M.L.M. is entered (SYS 4, etc.), the registers are automatically displayed. These registers are, from left to right, the:

Program Counter	(PC)
Interrupt Request vector	(IRQ)
Status Register	(SR)
Accumulator	(AC)
X Register	(XR)
Y Register	(YR)
Stack Pointer	(SP)

All of these registers, except IRQ, are contained inside the microprocessor. They have NO equivalent PEEK address and are only accessible through machine language. However, the M.L.M allows screen editing here so you can change the values of the registers with the cursor.

The Program Counter (PC)

The PC register always shows the address of the next instruction that will be executed. Notice that if you enter with SYS4, the PC will show 0005.

The IRQ Vector

Every 60th of a second the PET performs an interrupt. During the interrupt, the PET flashes the cursor, updates the clock, checks the stop key and scans the keyboard. Once again, all this is done by a machine code routine resident in ROM. The IRQ vector designates the starting address of this routine. You can change the IRQ vector (using the cursor), but the change does not take place until a 'G' command is given. Also, exit to BASIC ('X') resets the IRQ vector back to the original contents.

The Status Register (SR)

The Status Register (sometimes called the 'P' register) is used to indicate the status of the microprocessor. Each bit of this register is used to represent some condition in the processor as a result of the events leading up to its' interrogation.

Bit 0 - C - Carry Flag
1 - Z - Zero Flag
2 - I - Interrupt Disable
3 - D - Decimal Mode
4 - B - Break Command
5 - - Not used
6 - V - Overflow Flag
7 - N - Sign Bit

The Accumulator (AC)

The Accumulator is your main work register in the microprocessor. One can bring data into the accumulator and transfer it elsewhere in memory or perform operations on it such as AND, OR, ADC, etc.

The X and Y Registers (XR & YR)

These registers are auxillary registers. Although they can be used for data transfers, they are more often used as index registers for the accumulator.

The Stack and the Stack Pointer

The stack is an area in RAM used by the microprocessor to store registers temporarily while another task is performed that requires the use of those same registers. The stack is located from $0100 to 01FF. That's 255 bytes of the total 1K that PET takes away from you on power up. This area is critical for proper machine operation and no POKEs should be made here. The stack is often used to store return addresses for subroutines AND interrupts. All registers are also pushed on the stack before servicing an interrupt and pulled off the stack back into their respective registers at the end of an interrupt. Several arithmetic routines also use the stack extensively.

The Stack Pointer is used as an index pointer into the stack area. As the 6502 throws stuff onto the stack, it must keep track of which byte it can use next. The SP starts at $FF and decrements each time data is pushed on the stack. This means that the first byte onto the stack would occupy location $01FF. The SP would decrement by 1 and the next byte on would go into $01FE, and so on. As these bytes are removed from the stack, SP is first incremented, and the processor gets the byte contained in location $0100 + SP, or more formally, $0100,SP . Afterwards the Stack Pointer is conveniently left pointing at the next available location. (For more info on the stack, see Transactor #1, Vol3, pg5)

M - Display Memory

The 'M' command can be used to display any part of memory in the entire 64K range. Like the registers, this is also in hexadecimal. Syntax for the command is 'M,[start address],[end address]'. For example:

```
     PC  IRQ  SR AC XR YR SP
.;  0005 E455 30 00 5E 04 F8
.M,0400,0401
.:  0400 00 00 00 AA AA AA AA AA
.
```

The M.L.M displays groups of eight bytes on a line. Unless your end address is an even multiple of 8, you will always be shown the remainder of the line. Screen editing is allowed here too. Simply move the cursor over to the byte. Changes occur when RETURN is hit. The address can also be altered to duplicate lines into other memory addresses (like duplicating a line of BASIC by changing the line number and hitting RETURN).

G - GO or Execute

This command is equivalent to a GOTO as opposed to GOSUB. Programs should end with a BRK (hex 00) rather than an RTS. Upon executing the BRK, the registers will be displayed and you're back in the monitor.

The 'G' command can be used two ways. Once a machine language program is entered, a 'G' command followed by a hex address will start executing memory at the address specified.

If no address is given, execution will begin at the address shown by the Program Counter (PC). On a BRK, the PC will be left pointing at the address following the BRK instruction. After examining the registers, another 'G' command with no address would perform the equivalent of a CONT (continue).

X - Exit to BASIC

Quite simply, the 'X' command does a warm start to "READY." and you're back in BASIC.

L - Load

Unlike the BASIC LOAD command (or DLOAD), a monitor Load brings in files without changing any of the pointers that support BASIC text. This can be useful when you have a BASIC program already loaded and you would like to bring in a machine language utility.

BASIC programs can also be loaded with 'L', but pointers would not be set correctly and proper operation would be hampered. For example, NEW the machine and use the M.L.M. to load a BASIC program. You can LIST the program, but RUNning

the program will produce unpredictable results. LIST the program again and you'll notice your BASIC text has been clobbered. This is because the Start of Variables Pointer was not set properly to the end of BASIC but rather remained set pointing at the beginning of BASIC. As variables get set from within the program, BASIC begins building a variables table according to the address specified by the start of variables pointer. If this pointer is at the start of BASIC, whammo!... the variables table gets built right on top of your BASIC.

Syntax for the command is:

'L [filename],[device #]'

A drive number preceding the filename is optional and the device # must be in 2 digit hexadecimal. For example:

```
    PC  IRQ  SR AC XR YR SP
.;  0005 E455 30 00 5E 04 F8
.L "1:some program",08
searching for some program
loading
.
```

S - Save

Probably the most important of the M.L.M. commands. The 'S' function can be used to store any part of memory from zero page, cassette buffers, BASIC text, the screen, right up through BASIC ROM and the operating system. This command is most appreciated when a crash occurs that clobbers the BASIC operating system. Such situations are usually caused by a faulty machine code subroutine that you might be testing. When the BASIC SAVE command returns ?SYNTAX ERROR, it's nice to know that you can salvage the program from the M.L.M. Syntax is:

'S [drive#]:[filename],[device#],[start addr],[end addr+1]'

Due to a small bug, add 1 to the end address chosen. The byte at the resultant end address is not saved.

```
    PC  IRQ  SR AC XR YR SP
.;  0005 E455 30 00 5E 04 F8
.S "1:screen contents",08,8000,9000
.
```

The above would store an image of the screen out to a disk PRG file. Of course the monitor display would be stored too, but this would be impossible with an ordinary BASIC SAVE.

There is one exception when dealing with cassette tape. Due to the tape routines and format, the cassette will not accept addresses above $7FFF, or more precisely, addresses with bit 15 set to 1. Since the start address is stored in the file header, a save of memory above $7FFF will not be properly sent to the cassette. However, I can't think of a

single reason why this would be necessary since there's not much you could do with it on tape.

Summary

With this information in hand, you'll probably find yourself more intrigued by the concept of "machine language" (that is if you're not already). As mentioned earlier, there are several monitor extension programs that add some extremely useful commands to the built-in M.L.M. Commands like dissassemble, single step, hunt, transfer, interrogate, and assemble make life a lot easier. A little practice and you'll find that machine language is not as tough as it's made out to be. With many sophisticated assemblers now available, you might even find machine language easier than BASIC!

Features Of
The 6845 Video Controller

Jim Bottom
Control Microsystems

All Commodore machines that have 12" monitors (8032, fat 4032, 8096, and SuperPET) employ a device known as the 6845 video controller to generate the video signals. Some come with a 6545 but they're both the same.

The chip has 18 programmable registers that are accessed through 2 memory locations at $E880 and $E881 (59520 and 59521). To read or write a register, the number of the register (0-17) is stored in 59520 and that register can then be accessed through 59521. This method saves considerably on address space.

The device is used to control such functions as character height, cursor size, horizontal position, vertical resolution, etc. Here is a summary of the registers and their functions:

R0 - Horizontal Total Register. Horizontal frequency equalling the total of displayed plus non-displayed "character time units" minus 1.

R1 - Horizontal Displayed Register. Number of displayed characters per horizontal line.

R2 - Horizontal Sync Position. Controls horizontal positioning.

R3 - Horizontal Sync Width. 4 bits which control the width of the horizontal sync pulse.

R4 - Vertical Total. The vertical frequency is controlled by R4 and R5.

R6 - Vertical Displayed. Number of displayed character rows on the video.

R7 - Vertical Sync Position. Controls vertical positioning.

R8 - Interlace Mode. 2 bits which determine wether interlaced or non-interlaced mode is employed.

R9 - Maximum Scan Line Address. 5 bits determining the number of scan lines per character row including spaces.

R10 - Cursor Start. 7 bits for cursor start scan line and blink rate.

R11 - Cursor End. 5 bits for cursor end scan line.

R12 - Start Address. R12 and R13 control the first address put out as a refresh address after vertical blanking. R12 is the low 8 bits and R13 is the hi 6 bits of the address.

R14 - Cursor Register. This 14 bit register stores the cursor location. R14 is the 8 low and R15 is the 6 hi bits.

R16 - Light Pen. 14 bits of R16 and R17 store the contents

All registers are write only with the exception of R14, R15, R16 and R17. This means the registers 0-13 can only be POKEd. PEEKing these registers will return invalid results.

Interesting Effects

In the early days, it was possible to blank the screen of the 2001 PET. This was useful for visual effects. BASIC 2 machines omitted this capability which was unfortunate. Now it's possible again! The following program demonstrates screen blanking for 12" monitor machines:

```
10 FOR J=1 TO 2000 : PRINT "*"; : NEXT : REM FILL SCRN
20 POKE 59520, 1 :              REM HORIZ DISPLAY REG
30 POKE 59521, 0 :              REM NO WIDTH = NO DISPLAY
40 FOR J=1 TO 2000 : NEXT :     REM DELAY
50 POKE 59521, 40 :             REM VIDEO BACK ON
```

Notice that the register offset (59520) need not be set again to access the same register as before. Default display width for 40 and 80 columns screens is "40".

The horizontal and vertical positioning of the display area can be altered by POKing different values into R2 and R7. R2 default is 41 and R7 default is 29.

The number of displayable character columns may be modified by POKing different values into R1. R1 default is 40. Likewise, the number of displayable character rows can be changed using R6. R6 default is 25.

The machine makes use of the controller for things like windowing, and text/graphic modes. Several other effects could certainly be achieved for use in games and other applications. The combinations and permutations are virtually endless!

Editor's Note

A word of caution. Uneducated experimentation with the 6845/6545 can potentially crash the computer. The effects involving the registers that Jim talks about are all safe to play with but other registers should be left alone unless you know what you're doing.

Getting Usable Video
Signals From 12" Monitor PET/CBMs

Jim Law
Batteries Included

This program works with any "fat" Commodore computer (8032, 12" 4032, 8096, and SuperPET). It re-configures the video controller to produce signals which more closely approximate the standard video sync signal frequencies. For those wishing to use an external video adapter to display the PET screen on a video monitor, this program can save having to modify the monitor.

The PET screen remains completely readable, with only minor narrowing of the picture and the possibility of loosing part of the bottom line in text mode. However, this can be corrected with a slight adjustment* to the PET video section.

Video adapters formerly used with 9" screen machines will not work directly with the new 12" machines as the polarity of the video out and horizontal sync signals have been reversed at the User Port. Correct this with:

```
POKE 59520, 12 : POKE 59521, 0
```

This will give the desired effect on the external monitor but the PET screen will be inverted (but still readable). To get back to normal, POKE 59520, 12 : POKE 59521, 16

Some video interfaces for 9" machines have a horizontal sync position control and thus may accept the inverted pulses. People have used this system and it saved them having to obtain new interfaces.

Video adapters for 12" screen machines are now becoming more readily available and may be used with this program to produce an external picture which might otherwise be unable to "sync" to the strange signal frequencies coming from a 12" machine without this program.

A universal video adapter is available from Batteries Included in Toronto (416 596 1405) as well as "Video+"; a ROM which alters the signals at power-up and eliminates the need for this program. Video+ replaces the ROM at UD7 and does not interfere with normal operation.

```
100 DATA 0, 59, 2, 47, 4, 26, 5, 8, 7, 25, 9, 9
110 DATA 0, 59, 2, 47, 4, 33, 5, 6, 7, 30, 9, 7
120 FOR J=1 TO 6 : READ A, B
130 POKE 59520, A
140 POKE 59521, B
150 NEXT J
160 POKE 59468, 14 : PRINT CHR$(14)
170 PRINT "HIT 'STOP' FOR TEXT MODE CONFIGURATION"
180 GET A$ : IF A$="" THEN 90
190 FOR J=1 TO 6 : READ A, B
200 POKE 59520, A
210 POKE 59521, B
220 NEXT J
230 POKE 59468, 12 : PRINT CHR$(142)
240 PRINT "HIT 'STOP' FOR GRAPHICS MODE CONFIGURATION"
250 GET A$ : IF A$="" THEN 250
260 RESTORE : GOTO 120
```

PRINT-AT Routine

Jacques Lebrun
Lennoxville, Quebec

Ever had to print at random locations on the screen? Most programmers probably have. The only way provided by the BASIC interpreter is to use cursor down, up, left and right enclosed within quotes in print statements. This could be impractical for some programs which constantly print something at different places on the screen.

Here is a cure: a machine language PRINT-AT routine that allows a BASIC program to print anywhere on the screen with a simple SYS command. The routine is location independent which means that it can execute well no matter its location in memory. The best place is probably the first cassette buffer for those who never use cassettes. The routine is for BASIC 4.0 but could be modified for other BASICs

The routine can be used two ways. The first one is to position the cursor only. This allows further PRINTs, INPUTs and GETs to be done at that new location on the screen. Format is:

```
SYS ADDR,ROW,COL
```

The second way positions the cursor and prints whatever text follows the SYS. Its format is:

```
SYS ADDR,ROW,COL,TEXT
```

'ADDR' is the starting address of the routine, 'ROW' and 'COL' are two numeric expressions. 'ROW' ranges from 0 to 24 and 'COL' ranges from 0 to 79. 'TEXT' can be any numbers of string, integer or floating-point expressions seperated by commas or semi-colons just as a normal PRINT statement.

```
100 REM *** BASIC LOADER FOR PRINT-AT ROUTINE ***
110 REM *** SET VARIABLE AD TO DESIRED STARTING ADDRESS ***
120 AD=634
130 FOR I=AD TO AD+41
140 READ A
150 POKE I,A
160 NEXT I
170 END
180 DATA  32, 245, 190,  32, 212, 200, 224,  25
190 DATA 176,  28, 134, 216,  32, 111, 224,  32
200 DATA 245, 190,  32, 212, 200, 224,  80, 176
210 DATA  13, 134, 198,  32, 118,   0, 240,   9
220 DATA  32, 245, 190,  76, 168, 186,  76, 115
230 DATA 195,  96
```

The Print Mint

Jim Butterfield, Toronto

The usual point of programs is that they produce output. The normal way of producing output is by using the PRINT command or its cousin, PRINT#. We can abbreviate PRINT with a question mark (?), but oddly enough, PRINT# can't be shortened that way: typing ?# will produce a program line that lists as PRINT#, but doesn't work.

Other Ways.

We can generate output without using PRINT. It's not always good practice, but we can POKE to the screen memory area. This can be good for graphic games and animations, but there are several bonus things we get by using PRINT. First, PRINT keeps track of the line position for us, and starts a new line as necessary. Secondly, PRINT doesn't limit output to the size of the screen: when the screen fills up, scrolling is automatic. Finally, and most important, PRINT can easily be changed to PRINT# to allow output to be directed to other devices such as printer, modem, disk or cassette tape. In contrast, screen POKEs are absolutely limited to the size of the screen, and can't be easily redirected anywhere else.

Punctuation.

If you say 'PRINT X' you will print the value of X and start a new line. The absence of punctuation at the end of the PRINT command signifies, "That's the whole thing; print it and wrap the line up." In contrast, if you say 'PRINT X;' you will print the value of X but you won't go to the next line: the invisible cursor will wait behind the printed value. This sounds a little backwards: you do something extra if you have no punctuation, but you do nothing if you have a semicolon.

There's one other form of "formatting" punctuation: the comma. If you type 'PRINT X,' you will print the value of X and then skip ahead to the next "column". Columns are considered to start at positions 11, 21, 31 and so on up to position 71. They exist only on the screen; saying 'PRINT#4, X,' to send to printer or other device won't set up columns properly. The comma will produce quick and convenient output to the screen, but it may be a bad habit since you can't use it anywhere else.

We can use this punctuation within a Basic PRINT statement as well as at the end. 'PRINT A;B$;C;' will generate the values of variable A, string B$, and variable C one behind the other and will leave the cursor positioned behind the value of C.

Neat Input.

We can use this punctuation to generate prompting for INPUT statements. For example, if we wanted to add ten numbers, we might code:

```
100 PRINT "INPUT EACH NUMBER:"
110 FOR J=1 TO 10
120  PRINT J;
130  INPUT X
140  T=T+X
150 NEXT J
160 PRINT "TOTAL IS";T
```

We prompt for the ten numbers with 1? ... 2? ... 3? ... and so on. The prompting number is printed by line 120 - J is stepping from 1 to 10 - and the question mark from the INPUT statement appears behind it because line 120 ends in a semicolon; after printing the number we wait on the same line so that the question mark will appear there. Question: What would happen if line 120 ended with a comma instead of a semicolon? Try it and see.

Number Formats.

Numbers are printed in a special format. First, there is either a space for positive numbers or a minus sign for negative numbers. Then the number appears, as many digits as required plus a decimal point if needed and perhaps even "E" notation. (Never heard of E notation? Try PRINT 3E2 and see if you can figure it out). Finally, the number is followed by a cursor-right on the screen.

This seems at first to give you two spaces between numbers, but there are one or two fine points that are useful to know. If you type PRINT 2;3;4 you will see two spaces appear between each set of digits. Now try this: Type a bunch of x characters over the answer (a row of xxxxxxx...) and then cursor back to the PRINT statement and press RETURN again. Some of the x's don't go away; that's because a cursor-right skips over that part of the screen without writing there.

There are a couple of ways to eliminate this difficulty if it bothers you. If you change a value to a string before printing, the cursor-right won't be performed. You could type PRINT STR$(2);STR$(3);STR$(4) - the same numbers will print with at least part of the problem solved. If you happen to have an 80-column or Fat-40 4.0 system, you may type: PRINT CHR$(16);CHR$(22);2;3;4 and you'll discover the problem is solved quite elegantly.

Here's another exception to the two-spaces rule: Type PRINT 2;-3;-4;5 and look at the result. The minus signs take up one of the two positions, and now there's only one space between some numbers.

Summary.

PRINT is handy and versatile. It takes a little while to get used to the formatting of the PRINT statement, but you'll soon have good control over your output.

There are some fascinating things you can PRINT which cause the screen to do unusual things. More about them another time.

Programming Tips

Paul Higginbottom
Commodore Canada

First I would like to answer some frequently raised questions about screen formatting of data, and then take a look at a few techniques to make programs smaller and more elegant.

Number Juggling

Formatting numbers on the screen can cause problems when the TAB function is used. If a numbers are to be printed in columns, then it would be nice to ensure that the decimal point of the numbers always line up in the column.

For example, if the number is simply TABbed onto the screen, and the number is a "0", then it will appear at the left hand side of the column, which doesn't look very smart. It is therefore necessary to use the length of the number (ie. the number of digits including decimal points) to drive the TAB expression. A number has a leading space and a trailing cursor right which needs to be taken into consideration. The LEN function counts the number of characters in a string. In order to use LEN it is first necessary to convert the number into a string using STR$. The number of characters in the number is given by:

```
X = LEN ( STR$ (A) ) - 1
```

...where A is the number. The trailing cursor right is ignored by STR$ but 1 is subtracted to take account of the leading space. So now, taking the above example, if we were to TAB(10-X) we would be in business, right? No, not quite yet.

There are a couple more possibilities that should be considered. Sometimes it is desirable to tack leading zeroes onto integer numbers (ie. to display "0038" rather than "38"). With decimal numbers, we may want a specified number of digits following the decimal place, followed by trailing zeroes which the PET does not do for us. All of this will affect the positioning of numbers output to the screen.

Without question, the easiest way to handle number formatting is to first turn the number into a string:

```
A$ = MID$ ( STR$ (A), 2 )
```

STR$ converts "A" to a string and the MID$ function is used to take the 2nd character onwards, thus removing the leading space. To add leading zeroes we use the RIGHT$ function:

```
A$ = RIGHT$ ("000000" + A$, 4 )
```

Assuming "A" is an integer, the above will produce a 4 character string. The number of leading zeroes will be dependent on the size of "A" (ie "16" produces "0016" and "1024" stays "1024"). For compactness, all of the above could have been done with:

```
A$ = RIGHT$ ("000000"+ MID$ ( STR$(A),2), 4)
```

Rounding decimal numbers and adding trailing zeroes is a little trickier. First we must decide how many significant digits are to follow the decimal place. The following is an example for 2 significant digit truncating:

```
A = 1035.55534
A = INT (A * 100) / 100
PRINT A
 1035.55
```

The above merely moves the number two places to the left, chops off the fractional part, and then moves it back two places to the right. However, this is not rounding but rather truncating, which is not the same. For rounding, we must first decide what degree of rounding is desired. Most often, numbers are rounded to 2 decimal places, or "to the penny". Our example then becomes, simply:

```
A = 1035.55534
A = INT (A * 100 + .5) / 100
PRINT A
 1035.56
```

The same result would be accomplished by first adding .005 to A and multiplying that by 100, but BASIC is more accurate at decimal arithmetic that lies closer to the decimal point.

Numbers that end up with 1 decimal or less will need trailing zeroes. Once again, this can be done with a string manipulation. For example:

```
A = 1035.59534
A = INT ( ABS(A) * 100 + .5) / 100)
PRINT A
 1035.6
S$ = CHR$ (32 - (V<0) * 13)
A$ = MID$ ( STR$ (INT(A)) ,2)
DP = INT ((A - INT(A)) * 100 + .5)
A$ = A$ + "." + RIGHT$ ("00" + MID$ (STR$(DP) ,2) ,2)
A$ = RIGHT$ ("          " + S$ + A$ ,10)
PRINT A$
 1035.60
```

This probably looks sort of clumsy but it's designed to do everything; trailing zeroes; pos. and neg. rounding; and decimal point alligning. Notice that we've taken the ABSolute value of A before entering the routine. S$ will be either a space or a minus sign. Then we grab the INTeger part of A into A$. DP is used to take the decimal part of A, and round it on the left of the decimal point. Next we build it all together by taking the integer part, adding our own decimal point, followed by the decimal part. As usuall, the leading space is stripped off DP with MID$. "00" is added to this (remember, DP could have a value of say 04 or an even 0) and RIGHT$ comma 2 gives us our 2 decimal places. Lastly we add some leading spaces (which could also be zeroes) and then we stick the sign S$ of the front.

Easy, right? Well, you might have done it differently but this was a string juggling exercise for the practice. For a formatting routine that's even more cryptic than this one, see Jim Butterfield's PRINT USING in Transactor #1, Volume 3.

Variable Flip-Flop

Programs can be shortened a great deal with a little thought and an active imagination. For example, it is often necessary to set a flag if a condition is met or to compliment the flag. One might code:

```
1200 IF FLAG=0 THEN FLAG=1 : GOTO 1220
1210 FLAG=0
1220 ...
```

On consideration, the statement

```
1200 FLAG = 1 - FLAG
1210 ...
```

will be seen to have the same effect.

Screen Codes To ASCII

This program is a screen dump routine which makes it possible to copy the contents of the screen onto the printer. This subroutine has been presented several times but we'll be looking at technique as opposed to operation.

```
5000 OPEN 4, 4
5010 FOR J = 0 TO 999
5020 P = PEEK (32768 + J)
5030 GOSUB 5500
5040 IF P< 64 THEN P=P+ 64 : GOTO 5090
5050 IF P<126 THEN P=P+128 : GOTO 5090
5060 IF P<128 THEN P=P+ 64 : GOTO 5090
5070 IF P<191 THEN P=P- 64 : GOTO 5090
5080 IF P=255 THEN P = 191 : GOTO 5090
5090 PRINT #4, CHR$ (P);
5100 X=X+1 : IF X=40 THEN PRINT#4 : X=0 : F=0
5110 NEXT
5120 CLOSE 4
5130 RETURN
5500 REM REVERSE FLAG
5510 IF F=1 AND P>127 THEN 5550
5520 IF P>127 THEN F=1 : PRINT#4,"[RVS]"; : GOTO 5550
5530 IF F=0 AND P<127 THEN
5540 IF P<127 THEN F=0 : PRINT#4,"[RVS OFF]";:GOTO5550
5550 RETURN
```

As you can see, the routine begins PEEKing the screen into P. The subroutine at 5500 deals with reverse field characters. Then P is converted to its corresponding CHR$ value and it's sent to the printer. Lines 5010 and 5100 are set up for 40 columns but this can easily be changed to 80.

Now let's have a closer look at those five nasty IF statements. If we look at the differences between PEEK/POKE codes and ASCII, it becomes apparent that only the 3 most significant bits (bits 5 [32], 6 [64], and 7 [128]) are changed. "Aha, a bit of boolean algebra will solve this problem!" Using the OR and AND functions, it is possible to make the conversion with just one line! Thus if P=PEEK(32768), the top left corner of the screen, then:

```
C = (P AND 127) OR ((P AND 64)*2) OR ((64-P AND 32)*2)
```

where C is the corresponding ASCII character. Here's the new program:

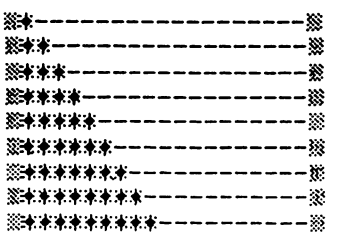
```
5000 OPEN 4, 4 : T=40 : S=32768
5010 FOR J = S TO 33767 : P=PEEK(J)
5020 R$ = CHR$(146-(PAND128))
5030 R$=LEFT$(R$,-(P>127ANDPEEK(J-1-(J=S))<128ORP<128AND
     PEEK(J-1-(J=S))>127))
5040 P=(P AND 127)OR((P AND 64)*2)OR((64-P AND 32)*2)
     -((PAND127)=0)*64
5050 PRINT#4, R$;CHR$ (P);
5060 IF (J-32807)/T = INT((J-32807)/T) THEN PRINT#4
5070 NEXT : CLOSE4
```

The extra little bit at the end of 5040 takes catre of "@" signs that have a screen code of zero. Lines 5020 and 5030 generate a RVS ON or RVS OFF character and then decide wether to send it or not (5030). This is dependent on the field of the last character. The routine has one bug though; if quotes are printed to the printer, any RVS or RVS OFF characters sent will appear literally. If you expect there may be quotes, you'll need to modify so that a CHR$(141); is sent to do a carriage return with no line feed. Then you'll need to position back to where you left off.

This routine is dreadfully slow but it was meant to be a exercise in boolean algebra. Several screen print utilities have been released that are in machine code. (KEYPRINT, Compute! and TPUG Club Library)

Abacus

My final program has no application apart from fun. It is a pseudo abacus or bead counter.



```
100 POKE 59468, 14 : PRINT "[CLR] ABACUS" : Q=12 : D=9
110 E=102 : L=64 : B=81 : R=13 : X=14 : S=1
120 FOR I=1 TO 9 : PRINT CHR$(230);
130 FOR J=1 TO I : PRINT CHR$(209); : NEXT
140 FOR J=I TO22 : PRINT CHR$(192); : NEXT
150 PRINT CHR$(230) : NEXT I
160 FOR I=1 TO 9 : P(I)=33179-I*40 : NEXT  ;FOR 40 COL
160 FOR I=1 TO 9 : P(I)=33579-I*80 : NEXT  ;FOR 80 COL
170 C=S :    REM SETS ROW TO 1
180 P=P(C) : REM SET SCAN LINE POSITION
190 IF PEEK(P)=L THEN P=P-S : GOTO 190
200 IF PEEK(P)=E THEN 230
210 POKE P, L : P=P+S : POKE P, B : IF PEEK(P+S)=L
    THEN 210
220 GOTO 170
230 T=D-C : FOR I=S TO X+C-S : POKE P(C)+R-I, L :
    POKE P(C)+Q-I-T, B : NEXT : C=C+S : GOTO 180
```

Using variables instead of constants (100-110) gives a faster execution time:

```
E = Edge POKE code for CHR$(230)
    the shade character
B = Bead code
S = Starting row (1 being bottom)
L = Line POKE code (abacus bars)
```

Lines 120-150 draw the initial display. The next job is to create an array with the poke addresses of the mid points of each row. The array must match the position of the initial display. If you want to move it, you'll also have to modify line 160.

Lines 170 onward do the rest. The routine uses a scanning technique that scans from the middle of the bottom row moving left (190) until it finds something that is not a line. If it is a bead then line 210 pushes that bead along the bar until that bead hits something that is not the bar (either the right hand edge or another bead). If it finds the left hand edge before finding a bead (200) then it knows that all of the beads have been moved over to the right. Now it moves them all back and pushes one over to the right on the next row up (240, 180 & 190).

One thing I found is that the beads move very fast. If you want to slow them down, add a delay loop in 190:

```
190 IF PEEK(P)=L THEN P=P-S : FOR I=1 TO 20 : NEXT :
    GOTO 190
```

I will leave you with a final puzzle - how long will it take before the top bead moves?

Paul Higginbottom
Commodore Canada

Power-Up Activity

When the VIC-20 is powered on, it looks for a sequence of bytes in a place in memory, which will tell the computer whether or not there is a cartridge plugged in. If this pattern of bytes is not present, the VIC goes about its usual chores, and comes up with the familiar message:

```
**** CBM BASIC V2 ****

3583 BYTES FREE        (may be a different amount)

READY.
```

The Sequence of Bytes

The sequence of bytes the VIC-20 is looking for are located from $A004 to $A008. They should contain:

Hex Address	Hex Contents	CBM ASCII Equivalent
$A004	$41	"a"
$A005	$30	"0"
$A006	$C3	"C"
$A007	$C2	"B"
$A008	$CD	"M"

To recap then, "a0CBM" is the sequence. If the VIC does see this sequence, what does it do? Well, that's what the first four bytes from $A000 to $A003 are for. If the VIC sees the sequence, it will hump indirectly through $A000, ie. using the vector at $A000 as a power up address. In laymans terms, this means wherever your code is that you want to execute at power-up, the address of it should be stored in "low, high" format at $A000 and $A001. This will be known as the "cartridge cold start vector".

That still leaves $A002 and $A003. They form another vector called the "cartridge warm start vector". This will be vectored through whenever an "NMI" interrupt occurs. For those that are not aware of this, the RESTORE key actually generates an NMI interrupt, and so it may be used to re-start a game, or better still if pressed while holding down another key simultaneously (usually RUN/STOP) to avoid accidental re-starts.

An example in assembly coding might be:

```
 *=$A000
;
.WORD COLD ;LOW, HIGH VECTOR TO FIRST INSTRUCTION
.WORD WARM ;LOW, HIGH VECTOR ON "WARM START"
;
.BYT $41, $30, $C2, $C3, $CD
;
;IGNORE NMI INTERRUPTS
;
WARM PLA
     TAX
     PLA
     TAY
     PLA
     RTI
;
COLD  <- CARTRIDGE CODE STARTS HERE.
;
```

BASIC Programs in a Cartridge

It is inadvisable to try to "RUN" BASIC programs out of a cartridge. A compromise would be to do the usual power-up activities (ie. test RAM, etc.) and then "move" the BASIC program from the cartridge to RAM, and execute (probably easiest to do by putting a RUN<RETURN> in the keyboard buffer). It is also inadvisable to put BASIC in a cartridge anyway because of the hazard of someone putting the code onto a more "transportable" medium such as tape or disk. This can be partially avoided by disabling the RUN/STOP key, and the RESTORE key by using the WARM start vector, and changing the IRQ vector into the cartridge.

Cartridge Standards

Here are some standards that have been laid out for operation of cartridge software:

1. Users should have the opportunity to adjust the screen before commencing; horizontal adjust ($9000) using the cursor left/right key and vertical adjust ($9001) using cursor up/down.

2. Where applicable, games should have both keyboard and joystick/paddle control. The keys to use are:

```
                                    Up
                                   P
A  - "fire"              Left - L    ; - Right
                                   .
                                 Down
```

3. Also where applicable, F1 and F3 are used to choose 1 or 2 players. Afterwards, F1 is used to start play action. Provide about a 5 second delay between "Play Player 1" and "Play Player 2".

4. Function key 7 should flip interlace mode. Some TVs scan every raster "one after another". Other TVs scan every second raster, then go back and do the "in between" rasters. The VIC chip can provide for this by simply setting or unsetting bit 7 of $9000. The user will decide which looks best.

5. Finally, the combination of the RUN/STOP and RESTORE keys should re-boot the game, as if power were interrupted.

LEARN

Discover how easy it is for you to get useful results from your VIC.

Understanding Your VIC Volume 1: Basic programming uses a proven step-by-step approach to teach programming. It costs $11.95

A cassette tape with two demonstration programs from the book is available for $7.95. It will save you typing time and eliminate typing errors.

VIC Software
Easy to use program for you to create new characters for graphics or games. Cassette $9.95.

See your dealer or order direct
VISA/MC accepted
Money back guarantee
Please add $2 ($8 overseas)
for shipping and handling

TIS INC.

Total Information Services, Inc.
Box 921, Dept. CM
Los Alamos, NM 87544

Dealer inquiries invited.

A Little VIC Music. Jim Butterfield, Toronto

The following program plays music on the VIC. It's the simple round, Frere Jacques. The music is listenable, and the program is worth looking at, too.

You'll note that the three voices of VIC are different. Voice three is sharper, and is better for carrying the tune. Voice one is the softest.

Hope you don't mind my breaking up the listing with comments.

```
 90 REMARK:  FRERE JACQUES /JIM BUTTERFIELD /DECEMBER 81
```
This tells you who to blame.
```
100 DIM A(8)
```
Makes room for eight voices. How come? We only have three voices on the VIC and four "lines" in the song. Watch for the trick.
```
110 POKE 36878,8
```
Set volume (for maximum use ,15)
```
120 FOR A=5 TO 0 STEP -1
```
Here's our main loop. We're going to play the tune six times.
```
130 T=TI+S
140 IF TI<T GOTO 140
```
This waits for time "S" before allowing the program to continue. The time is measured in "jiffies": units of 1/60 second.
```
150 READ S, A(A+0), A(A+1), A(A+2), A(A+3)
```
Here comes the song data. It's taken from the DATA statements near the end of this program. We're reading the data into the table cleverly; this way, each voice "comes in" at the proper time.
```
160 POKE 36874,A(3) : POKE 36875,A(4) : POKE 36876,A(5)
```
Play the music! This puts the notes into the VICs playing electronics.
```
170 IF S<>0 GOTO 130
```
If there's no more music to play, variable S will become zero (from the DATA statement at line 1120) We may want to do it again, though.
```
180 RESTORE : NEXT A
```
RESTORE takes us back to the start of the data statements (line 1000) so that we can play it again if we wish. NEXT A takes us back for the six repeats.
```
190 POKE 36878,0 : END
```
Turn down the volume and quit. The END statement isn't really needed here, but it's good practice.

The rest of the program is our DATA statements containing the music. It's set up with a timing value followed by the four "parts". By careful reading of the program, you may be able to work out how the different voices come in during the repeats (hint: the key to the trick is in lines 150 and 160).

```
1000 DATA 10, 195, 207, 215, 195
1010 DATA 10, 195, 207, 219, 195
1020 DATA 10, 201, 209, 215, 175
1030 DATA 10, 201, 209, 209, 175
1040 DATA 20, 207, 215, 207, 195
1050 DATA 20, 195, 215, 195,   0
1060 DATA 10, 195, 207, 215, 195
1070 DATA 10, 195, 207, 219, 195
1080 DATA 10, 201, 209, 215, 175
1090 DATA 10, 201, 209, 209, 175
1100 DATA 20, 207, 215, 207, 195
1110 DATA 20, 195, 215, 195, 195
1120 DATA  0,   0,   0,   0,   0
```

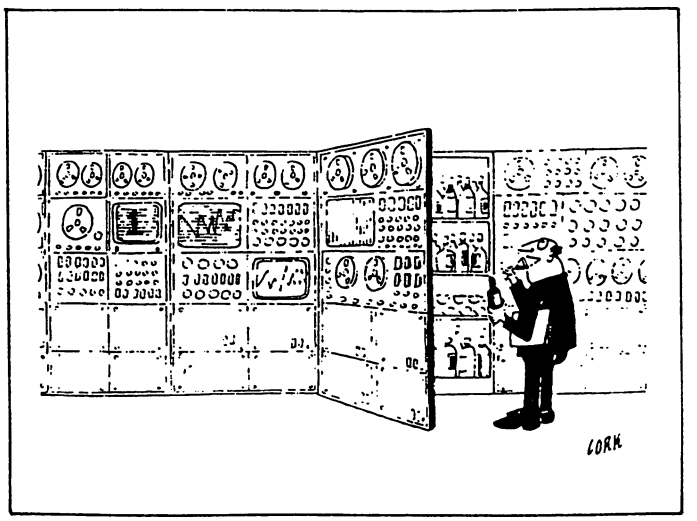
It's not very big, but it's interesting to see how the coding comes together. Check Appendix F of your VIC-20 Friendly Computer Guide and you'll see how to set up the notes. Write your own music. If you like programming you might want to try your hand at writing a program which allows DATA statements to be written in easier form. For example, line 1000 might be written as DATA 10,C,E,G,C .. but your program will need to be smart enough to catch the letters and translate them into the appropriate numbers.

Music doesn't have to stand by itself, of course. You could add it as an extra touch to games and animations. Looking at it the other way, you could add to the music - how about a "bouncing ball" program that lets you sing along with VIC?

You can get some nice effects from the VIC, although you'll never quite achieve orchestra quality sound. I can recall showing a group of users some simple music coding on the VIC. At one point, I played a simple rendition of "Dixie", and noticed a listener who had tears in his eyes. I was touched. I asked him, "Are you a Southerner?"

"No," he replied. "I'm a musician."

I guess you can't win 'em all.



Another Voice For The VIC-20

Andy Finkel
Commodore U.S.A.

Normally, your VIC has 4 musical voices... three music registers and a white noise register. But by connecting a small amplifier to the User Port, and doing a little programming, you can get another musical voice.

The User Port on the VIC is very similar to the PET User Port. This makes it easy to adapt some of the PETs music methods to the VIC-20.

Background - Adding Sound to Older PET/CBMs

Before Commodore introduced the CBM 8032 with a built-in speaker, most PET/CBM users had to develop their own means of getting their computers to squeek, hum, whistle, and sing. They came up with the idea of using the User Port to send square waves through an external amplifier/speaker combination. The shift register could be programmed through BASIC, giving a wide variety squeals, pops, sirens, etc.

Theory

Most music is made up of square waves of different amplitudes and frequencies. One of the functions of the 6522 chip is to generate square waves through the CB2 line. If we connect the CB2 line to a speaker, we will be able to hear the square waves generated by the VIC.

NOTE: Connecting a speaker directly to CB2 may damage your VIC and void your warranty. You must connect the speaker through an amplifier to protect the VIC. See Transactor #6 for a simple amplifier circuit.

BASIC Program Steps

1. Set the 6522 shift register to free running mode by typing:

```
POKE 37147, 16
```

2. Set the shift rate by typing:

```
POKE 37144, C
```

   ...where C is an integer from 0 to 255. C is the note to be played.

3. Load the shift register by typing:

```
POKE 37146, D
```

   ...where D = 15, 51 or 85 for a true square wave. This step also sets the octave for the note.

This step must be done last, since as soon as it is set, the VIC starts generating the square waves.

The frequency of the square wave can be found by the following formula:

$$FREQ = \frac{50000\ Hz}{(C+2)*(D1)}$$

where D1=8 when D=15
D1=4 when D=51
D1=2 when D=85

When you're in this mode, the VIC will not read or write to cassette. To restore normal operation, just type:

```
POKE 37147, 0
```

Here are some pre-calculated values for C:

Bb	= 251	B	= 124
C	= 237	C1	= 117
C#	= 224	C1#	= 111
D	= 211	D1	= 104
D#	= 199	D1#	= 99
E	= 188	E1	= 93
F	= 177	F1	= 88
F#	= 167	F1#	= 83
G	= 157	G1	= 78
G#	= 149	G1#	= 73
A	= 140	A1	= 69
A#	= 132		

The following short program demonstrates music using this method. By hitting a letter, a note will be played.

```
10 PRINT "MUSIC USING CB2"
20 REM A TO G IS ONE OCTAVE, SHIFT A TO G IS ANOTHER
30 PRINT "HIT + TO GO UP AN OCTAVE, - TO GO DOWN"
40 PRINT "[DN] USE ! TO EXIT."
50 POKE 37147, 16
60 DIM A(14)
70 FOR I=1 TO 14
80 READ A(I)
90 NEXT
100 DATA 124, 117, 104,  93,  88,  78,  69
110 DATA 251, 237, 211, 188, 177, 157, 140
200 GET A$ : IF A$="" THEN 200
210 IF A$="!" THEN POKE 37147, 0 : END
220 IF A$="+" THEN SF=SF-(SF<2) : GOTO 200
230 IF A$="-" THEN SF=SF+(SF>0) : GOTO 200
240 A=ASC(A$)-64+(ASC(A$)>192)*121
250 IF A>14 OR A<1 THEN 200
260 POKE 37144, A(A)
270 POKE 37146, -(SF=0)*15-(SF=1)*51-(SF=2)*85
280 GOTO 200
```

Joystick Control on The VIC-20

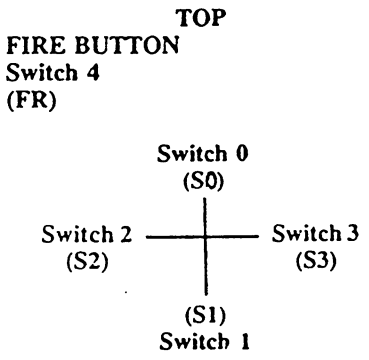
Andy Finkel
Commodore U.S.A.

Like all other input and output, the joysticks are controlled using the VIC's 6522 versatile interface adapters (VIAs). The 6522 is a versatile and complex device. Fortunately, it isn't necessary to delve deeply into the mysteries of the 6522 VIA to read the joysticks.

Each 6522 has two Input/Ouput ports, called port A and port B. Each of the ports has a control register attached called the DATA DIRECTION REGISTER (DDR). This highly important register controls the direction of the port. By using this register, you can use the port for input, output, or both at the same time. To set one bit of the port to output, set the corresponding bit of the Data Direction Register to a 1. To set a bit of the port for input, set the corresponding bit of the DDR to 0. For example, to set bit 7 of port A to input and the rest of the bits to output, POKE a 127 into the DDR for port A.

To read the joystick, one port (and one DDR) of each of the 6522 VIAs on the VIC must be used.

The joystick switches are arranged as follows:



Switch 0, switch 1, switch 2 and the Fire button can be read from VIA #1, which is located beginning at $9110. Switch 3 must be read from the other 6522 which is located at $9120.

Now, the key locations for the joystick are as follows:

Hex	Decimal	Purpose
9113	37139	Data Direction Register for I/O Port A on VIA #1
9111	37137	Output Register A Bit 2 - Joy switch 0 Bit 3 - Joy switch 1 Bit 4 - Joy switch 2 Bit 5 - Fire button
9122	37154	Data Direction Register for I/O Port B on VIA #2
9120	37152	Output Register B Bit 7 - Joy switch 3

To read the joystick inputs, you first set the ports to input mode by setting the DDR to 0. This can be done by a POKE. Then the value of the switches can be read with two PEEKs. Sounds easy, right? There is only one problem... VIA #2 is also used for reading the keyboard. Setting the DDR can disrupt the keyscan rather badly. So you have to make sure you restore the DDR to its original condition if you want to use the keyboard afterwards.

To make thing really easy, you can use the following program. Lines 10 to 40 are initialization. The rest of the program, beginning at line 9000, can be called as a subroutine whenever you want to read the joystick.

```
10 DIM JS (2,2) : POKE 37139, 0
15 DD=37154 : PA=37137 : PB=37152
20 FOR I=0 TO 2
25 FOR J=0 TO 2
30 READ JS (J, I)
35 NEXT J, I
40 DATA -23, -22, -21, -1, 0, 1, 21, 22, 23
50 GOSUB 9000
60 PRINT JS (X+1, Y+1)
70 GOTO 50
9000 POKE DD, 127
9010 S3=-((PEEK(PB) AND 128)=0)
9020 POKE DD, 255
9030 P=PEEK(PA)
9040 S1=-((P AND 8)=0)
9050 S2=-((P AND 16)=0)
9060 S0=((P AND 4)=0)
9070 FR=-((P AND 32)=0)
9080 X=S2+S3 : Y=S0+S1
9090 RETURN
```

The variables S0, S1, S2 and S3 will be 0 normally, and will be set to 1 (or -1) when the joystick is pointed in that direction. Two of the variables will be set to 1 on diagonal moves. FR will be 1 when the Fire button is pressed, and 0 otherwise.

The AND function is used to pick out one bit of the jpystick port. The bits are numbered from 7 (most significant bit) to 0 (least significant bit). By ANDing the 6522 port with a number whose value is a power of two, a single bit is selected (For example, to pick out bit 3, AND with 2, 3 or 8)

The JS array in the program is set up to make it easy to move around the screen using the joystick. The numbers in the DATA statement of line 40 can easily be changed for other purposes. For example, to decode the joystick in this pattern:

```
      TOP
       0
     7 ! 1
   6 - 8 - 2
     5 ! 3
       4
```

...the DATA statement should be changed to:

Computer Magic

Michael Tomczyk
Commodore U.S.A.

Writing programs for Commodore's VIC-20 is a lot like performing magic. The results are certainly astounding, and your friends are sure to be amazed!

Actually, computing isn't much different from magic if you're talking about illusions. For example, the VIC automatically tells you if you make a programming mistake by displaying an "error message" on the screen. That doesn't mean there's an "intellectual rabbit" hiding under the keyboard. It simply means the VIC-20 is a "logical" machine.

We're going to explore the VICs peculiar logic - and some magic too. This article will focus on elementary programming to show first-time computer owners how to COMPUTE, with secondary emphasis on hard-line programming. The philosophy is: "You don't have to know how to repair a car in order to drive one; likewise, you don't have to be a computer scientist to "drive" the VIC-20."

That's the beauty of Commodore's "friendly computer." It's easy to learn, fun to "drive", and you don't need a license (or PhD.) to use it.

Everyone likes to perform magic with their new computer, but doing these neat tricks the salesperson showed you in the store doesn't seem so easy when you get the thing home.

Here are some of the favourite programs of Commodore's "VIC Group". These programs are not only fun, but they incorporate some helpful computing techniques which you might want to mix, match, and experiment with. Most of these programs are explained in the VIC owners guide.

Before we begin, here's a quick refresher on how to enter a program into the VIC-20.

1. Type the program line-by-line as shown, including the line number.

2. Hit the RETURN key at the end of each line.

3. Type the word RUN and hit RETURN to make the program execute.

4. To stop a program which is "running", hit the RUN/STOP key.

5. You can RUN a program over and over by STOPping it and typing RUN (because the program stays in the VICs memory when you type it in).

6. Before typing in a new program, type the word NEW and hit RETURN to erase the old program.

7. If a program "hangs up", hold down the RUN/STOP key and hit the RESTORE key. This resets the VIC without losing the program.

VIC Trick #1: 255 Colours

```
10 FOR X=0 TO 255
20 POKE 36879, X
30 PRINT CHR$(147)
40 FOR T=1 TO 700 : NEXT T
50 NEXT X
```

This little program displays the VICs 255 screen and border colour combinations. It's useful because you can go through all 255 combinations step-by-step, and find the colour combination you like best for a particular program. The POKE command in line 20 is the key. RUN the program until you see a color combination you like. Hit the RUN/STOP key to freeze the colours. Now type PRINT X, and record the value of X for future reference. Type CONT and hit RETURN to continue the program where it left off. This is the lazy approach to choosing colours. The best way is to check page 37 or 134 in the VIC owners guide.

Vic Trick #2: The Rolling Screen Window

```
10 POKE 36867, 4
20 PRINT CHR$(147)
30 PRINT "YOUR MESSAGE HERE....."
40 FOR X=0 TO 120
50 POKE 36865, X
60 NEXT
70 GOTO 30
```

You can change the screen window of the VIC-20 by using some of the special POKE commands, which change the size and position of the VICs screen window. This little program makes use of these commands to make your messsage scroll downward across the screen. The message in line 30 should be 22 characters long. Try typing 22 hearts (hold down the SHIFT key and type S) instead of a message.

Vic Trick #3: The Seasick Program

```
10 PRINT CHR$(147) "SEASICK"
20 FOR L = 0 TO 6.28 STEP .1
30 POKE 36864, 5 + 4 * SIN(L)
40 POKE 36865, 27 + 4 * COS(L)
50 NEXT
60 GOTO 20
```

This program makes the screen move around... and around... and around. We call it the Seasick Program because that's how you might feel if you stare at it too long. The programming magic here is the VICs ability to move the screen horizontally and vertically using POKE statements.

Vic Trick #3: Drawing A High Resolution Circle

```
10 FOR S=7168 TO 7679 : POKE S, 0 : NEXT
20 POKE 36879, 8 : PRINT CHR$(147)
30 FOR S=7680 TO 8185 : POKE S, 160 : NEXT
40 POKE 36869, 255
50 FOR L=0 TO 7 : FOR M=0 TO 7
60 POKE 7680 + M*22 + L, L*8 + M
70 NEXT M, L
80 FOR X=0 TO 63
90 Y1=32 + SQR(64*X-X*X)
100 Y2=32 - SQR(64*X-X*X)
110 FOR Y=Y1 TO Y2 STEP Y2-Y1
120 CH=INT(X/8) * 8 + INT(Y/8)
130 RO=(Y/8 - INT(Y/8))*8
140 BY=7168 + 8*CH + RO
150 BS=7-(X-INT(X/8)*8)
160 POKE BY, PEEK(BY) OR (2 ↑ BS)
170 NEXT Y, X
180 GOTO 180 : REM FREEZE PROGRAM
```

This 18 line program looks like a lot... but it does a lot. This is our first program that actually draws something on the VIC-20 screen in high resolution, dot programmable graphics. Dot programmable graphics are different from VIC graphics, in that VIC graphics are made up of 8x8 dot blocks (64 dots per block). Dot programming lets you access each dot individually, and "draw" in high resolution. The mechanics of how to do this are discussed in the VIC-20 Programmers Reference Guide. But, if you really want to get into programmable graphics and plotting, we suggest the VIC-20 Super Expander Cartridge. This special cartridge gives you 3K of extra RAM and adds several new commands to VIC BASIC that let you plot individual points, lines, arcs and circles... and even let you "paint" closed figures on the screen in colour! The Super Expander also has built-in music writing commands and a special "music mode".

We hope this brief "magical" introduction to the VIC gives you some interesting programs to experiment with. If you have a particular topic you'd like us to discuss, please drop a line to:

VIC Magician
Commodore Magazine
681 Moore Road
King of Prussia, PA
19406

VIC Loader For PET/CBM

David A. Hook
Barrie, Ontario

Purpose:

As the VIC-20 becomes more popular, it's likely that users will be exchanging tapes with PET/CBM users. Many people may have a VIC at home, whereas their school or business is equipped with PET/CBM machines.

The VIC has several possible memory configurations, based on what is plugged into the expansion port. The "Start-of-Basic" may be at any of three locations (so far!). To allow for this, the VIC ROMs use a "relocating loader" to place the tape program into memory. Therefore, it merrily accepts BASIC programs saved on either PET/CBMs or other VICs, regardless of the machine used to SAVE it.

Not so for the program saved on the VIC that is subsequently loaded into a PET/CBM! BASIC programs are expected to be found at decimal 1025 ($0401). Only one of the VIC memory arrangements would have put it there (3K expansion cartridge).

After the LOAD, you may not find anything there when you do a LIST. A couple of tricks have appeared in print to adjust the PET "pointers", but these presumed VIC start-of-Basic at 4097 decimal ($1001). This only works if it was SAVEd with the "as-delivered" 5K unit. (POKE 4096,0:POKE 41,16:CLR was one of these).

If the above two things still didn't work, you could try POKE 4608,0:POKE 41,18:CLR. A LIST command would now probably show the VIC program, and allow you to modify the program.

However, a SAVE to a PET disk would leave you in the same dilemma next time.

Why not a loader program which takes account of these posibilities? It should load the VIC program into the normal PET BASIC area, without the above headaches.

A further incompatibility with the VIC is also overcome:

The VIC operating system allows you to save "program files" which are NOT relocated by the VIC on loading. Machine-language programs and user-defined character sets are two things that SHOULD NOT be parked at a variable address.

A special tape header is written by the VIC when this type is saved. The PET doesn't digest this, so any attempt to LOAD one of these will be ignored by the PET's operating system.

Jim Butterfield's "TINYMON" for VIC uses this "absolute load" feature for any SAVEs of memory. My adaptation of his Supermon (for VIC) also copies this technique.

This loader will accept such tapes into the PET, placing them at the same address and reporting start/end locations in hexadecimal.

Procedure:

You'll need to do this in three stages:

1) Type in the BASIC portion and save it to tape or disk.

2) Enter the machine-language monitor, typing a bunch of hexadecimal numbers. Save this part separately, too.

3) Combine the two, then SAVE the composite program.

I'll attempt to lead the way through the puzzle... it's not too long a process.

Step 1:

If you have Original ROMs (BASIC 1.0), you may retire early. This won't work. For all others, turn your machine off and back on again.

Type in the BASIC portion, by following the printed listing exactly. Don't leave out anything or insert any extra either. Those cursor movement mnemonics should be replaced by their corresponding characters (ie. CLR=Clear Screen, DN=Cursor Down, etc). When you are finished, '?FRE(0)' to check available memory. For 16K this should be 14809 bytes free, and for 32K, 31193 bytes free. Check carefully if you are more than a few different than this.

Do NOT attempt to RUN this yet. (If you cannot resist the urge to check it, put a 'REM' in front of the 'SYS' instruction first).

Save this program to tape or disk, as file name "VL.BAS". Verify it normally. (Did you remember to delete the 'REM' that you inserted?).

Step 2:

Consult the "hex dump" of the program, which accompanies this article.

Type 'SYS4' to enter the M.L. monitor. Don't be intimidated by the unfamiliar display... this won't take long.

Beside the ".", type the following:

```
M 0640 06F0      <RETURN>
```

The screen will fill with lots of numbers and letters. Change the values to match the data in the hex dump. Don't forget to hit 'RETURN' at the end of each line. A double-check may save later grief.

When this screen is done, type:

```
M 06F8 07B0      <RETURN>
```

Enter the correct values from the table, and double-check.

Now we are ready to save this part, so type:

```
S "0:VL4.ML",08,0640,07B8  (Drive #0 on disk)
S "VL4.ML",01,0640,07B8    (Tape #1)
```

BASIC 4.0 users can proceed to Step 3. Upgrade (BASIC 2.0) users need to make the following corrections. Type the instructions, then alter the bytes that are displayed to match those below:

```
M 0728 072F

.: 0728 20 56 F6 20 12 F8 20 0A

M 0738 073F

.: 0738 00 20 97 F4 D0 08 4C 6E

M 075E 076A

.: 075E 6A E7 A9 2D 20 D2 FF 20
.: 0766 97 E7 20 6A E7 B8 50 28

M 0797 079B

.: 0797 B9 F3 4C DD F3 A5 9D 48

M 07A0 07AF
.: 07A0 55 F8 A0 00 00 B1 D6 C9
.: 07A8 03 F0 03 4C B0 F5 4C BC
```

Now we are ready to save this part, so type:

```
S "0:VL2.ML",08,0640,07B8  (Drive #0 on disk)
S "VL2.ML",01,0640,07B8    (Tape #1)
```

Step 3:

Get back to BASIC, by typing:

X <RETURN>

Reload "VL.BAS" followed by the proper "VLn.ML" (n=2 or n=4). Save the composite program with a normal BASIC SAVE command.

Do not make any adjustments to the BASIC portion. It would move the machine language too... and goodbye forever!!!

Barring mistakes, you should be ready to make a go of it now.

Operation:

Type 'RUN' and the program will relocate the machine-language portion. It moves up to high-memory automatically, correcting the necessary pointers. BASIC will not interfere with its operation.

On the screen, the necessary 'SYS' address will be displayed. Copy this down, as it will permit you to disengage the routine with the same 'SYS' call.

The routine is now active. To load a VIC tape into the PET, type:

<V "FILE NAME" <RETURN>

The "<" must be in the first column of a screen line or the PET will ignore it. The "file name" is optional--if omitted, the load will be done on the first program found.

You will get the normal messages, ie. 'PRESS PLAY...', so follow normal procedure. When the 'READY' message appears and the cursor returns, you can LIST/edit/reSAVE just as if it were entered on the PET in the first place.

If the program was of the special, "absolute load" variety, the PET will load it in the same spot in memory it was SAVEd at (on the VIC). However, alongside the file name, the start and end address (in hex) will be printed. This will flag this type of load, and allow you to find it more easily in the PET memory.ln3; Both TINYMON and SUPERMON for the VIC save memory using this "absolute load" feature of the VIC-20. The PET would ignore the file, without this program.

That's about it. I was able to use this myself on VIXEL#1 (The Code Works) the day after it was finished. It's pretty handy to move VIC programs to PET disk, as a compact back-up medium.

David A. Hook
58 Steel Street
BARRIE, Ontario
L4M 2E9
(705) 726-8126

Editor's Note

If you're wondering why the source code for Daves' Vicloader doesn't match the hex dump, it's because they don't! The BASIC portion of the loader is immediately followed by Jim Butterfields' machine code relocator. After this comes Daves' code which still looks somewhat different from the source output since Jims' relocator demands a specific format. Perhaps JB will document his relocator in a future article so that other ML programmers may include it with new utilities.

Getting Acquainted With Your VIC20

Getting Acquainted With Your VIC20 by Tim Hartnell leads the reader, step by simple step, from the absolute basics of programming the VIC to writing complex, sophisticated programs. It thoroughly describes use of the sound, music and color graphics capabilities and illustrates the use of these functions in over 60 programs and games.

By following the comprehensive explanation given for each program and computer function, the reader will learn a great deal about the VIC, the Basic language and micro-computers in general.

Parents and teachers will find the section "VIC as a Teacher" a valuable aid in making the most effective use of the computer in the teaching/learning process.

This book is a worthwhile resource and will help the reader make the most of his computer. The reader will never feel quite the same about it after surviving a round of FRENZY, or listening to the VIC20 compose a 'symphony'.

Softbound, 132 pages, 5 1/2" x 8", $8.95; add $1.50 for shipping and handling.

creative computing

ATTN: Fiona
39 E. Hanover Avenue
Morris Plains, NJ 07950

Toll-free 800-631-8112
In NJ 201-540-0445

```
100 REM   LOAD VIC TAPES INTO PET
110 REM     FOR BASIC 4.0 ONLY
120 REM   AS OF FEBRUARY 21, 1982
130 REM
140 REM       (C) DAVID A. HOOK
150 REM        58 STEEL STREET
160 REM   BARRIE, ONTARIO, CANADA
170 REM   L4M 2E9    (705) 726-8126
180 REM
190 REM ALL COMMERCIAL RIGHTS RESERVED
200 REM
210 PRINT"[CLR RVS]"TAB(15)"VIC LOADER"
220 SYS1600
230 PRINT"[DN DN DN]- ACTIVATE OR CANCEL THE LOADER USING:"
240 SA=PEEK(52)+256*PEEK(53)
250 PRINTTAB(10)"[DN DN]SYS("SA")"
260 PRINT"[DN DN]- TO LOAD A VIC TAPE, TYPE:"
270 PRINT"[DN DN]<V "CHR$(34)"FILE NAME"CHR$(34)
280 PRINT"[DN DN]- FILE NAME IS OPTIONAL.
290 PRINT"[DN]- TYPE THE COMMAND AT COLUMN '0'."
```

```
.:  0640 A5 2A 85 1F A5 2B 85 20
.:  0648 A5 34 85 21 A5 35 85 22
.:  0650 A0 00 A5 1F D0 02 C6 20
.:  0658 C6 1F B1 1F D0 3C A5 1F
.:  0660 D0 02 C6 20 C6 1F B1 1F
.:  0668 F0 21 85 23 A5 1F D0 02
.:  0670 C6 20 C6 1F B1 1F 18 65
.:  0678 21 AA A5 23 65 22 48 A5
.:  0680 34 D0 02 C6 35 C6 34 68
.:  0688 91 34 8A 48 A5 34 D0 02
.:  0690 C6 35 C6 34 68 91 34 18
.:  0698 90 B6 C9 BF D0 ED A5 34
.:  06A0 85 30 A5 35 85 31 6C 34
.:  06A8 00 AA AA AA AA AA AA AA
.:  06B0 BF AE FE FF 00 E4 34 AD
.:  06B8 FF FF 00 E5 35 B0 0B 86
.:  06C0 34 86 30 AD FF FF 00 85
.:  06C8 35 85 31 A2 03 B5 78 48
.:  06D0 BD FA FF 00 95 78 68 9D
.:  06D8 FA FF 00 CA D0 F1 60 C9
.:  06E0 3C D0 08 48 A5 77 C9 00
.:  06E8 00 F0 08 68 C9 3A B0 EF
.:  06F0 4C 7D 00 00 20 70 00 00
```

```
.:  06F8 C9 56 D0 F1 A2 01 86 D4
.:  0700 CA 86 D1 86 9D A9 02 85
.:  0708 DB 20 70 00 00 AA F0 17
.:  0710 C9 22 D0 F6 A6 77 E8 86
.:  0718 DA 20 70 00 00 AA F0 08
.:  0720 C9 22 F0 04 E6 D1 D0 F2
.:  0728 20 95 F6 20 57 F8 20 49
.:  0730 F4 A5 D1 F0 0B 20 E7 FF
.:  0738 00 20 D6 F4 D0 08 4C AD
.:  0740 F5 20 E7 FF 00 F0 F8 A5
.:  0748 96 29 10 D0 4C E0 01 F0
.:  0750 1D E0 03 D0 DE BD 7B 02
.:  0758 95 FB CA 10 F8 20 17 D7
.:  0760 A9 2D 20 D2 FF 20 44 D7
.:  0768 20 17 D7 B8 50 28 AD 7D
.:  0770 02 38 ED 7B 02 AA AD 7E
.:  0778 02 ED 7C 02 A8 A5 28 8D
.:  0780 7E 02 A5 29 8D 7C 02 8A
.:  0788 18 6D 7B 02 8D 7D 02 98
.:  0790 6D 7C 02 8D 7E 02 20 F8
.:  0798 F3 4C 1C F4 A5 9D 48 20
.:  07A0 9A F8 A0 00 00 B1 D6 C9
.:  07A8 03 F0 03 4C EF F5 4C FB
.:  07B0 F5 4C 31 FF 00 08 FF 00
```

```
                ;
                ;****************************************
                ;*                                      *
                ;*            VICLOAD4.SRC5             *
                ;*    FOR LOADING VIC FILES IN 4.0 PETS *
                ;*        AS OF FEBRUARY 21, 1982       *
                ;*                                      *
                ;*     DAVID A. HOOK, 58 STEEL STREET   *
                ;*        BARRIE, ONTARIO  L4M 2E9      *
                ;*          CANADA (705) 726-8126       *
                ;*                                      *
                ;****************************************
                ;
                ;OP SYSTEM VARIABLES
                ;
7F08            CR      =   $0D     ; CARR. RETURN
7F08            QUOTE   =   $22     ; QUOTE CHAR
7F08            TXTTAB  =   $28     ; START OF BASIC
7F08            FRETOP  =   $30     ; STRINGS LOW LIMIT
7F08            MEMSIZ  =   $34     ; TOP OF MEMORY
7F08            CHRGET  =   $70     ; GET A CHARACTER
7F08            CHRGOT  =   $76     ; RE-GET LAST CHAR.
7F08            TXTPTR  =   $77     ; CURRENT TEXT POINTER
7F08            ST      =   $96     ; STATUS BYTE
7F08            VERCK   =   $9D     ;LOAD/VERIFY
7F08            FNLEN   =   $D1     ; FILENAME LENGTH
7F08            FA      =   $D4     ; DEVICE NUMBER
7F08            TBUF    =   $D6     ; POINTER TO TAPE BUFFER
7F08            FNADR   =   $DA     ; FILENAME ADDRESS PTR
7F08            TMP0    =   $FB     ; M.L.M. TEMP. POINTER
7F08            TAPE1   =   $027A   ;CASSETTE#1 BUFFER
                ;
                ;OP SYSTEM ROUTINES
                ;
7F08            WROA    =   $D717   ;WRITE ADDRESS FROM $FB,$FC
7F08            T2T2    =   $D744   ;SWAP TMP0 & TMP2
7F08            ZZZ     =   $F695   ;SET TAPE BUFFER POINTER
7F08            CSTE1   =   $F857   ;TAPE MSG
7F08            LD300   =   $F449   ;PRINT FILENAME
7F08            FAF     =   $F4D3   ;FIND SPECIFIC FILE
7F08            FAF1    =   $F4D6   ;FAF+3
7F08            FAH     =   $F5E5   ;FIND ANY FILE
7F08            FAH1    =   $F5EF   ;FAH+10 (CONTINUE)
7F08            FAH2    =   $F5FB   ;FAH+22 (ABS LOAD)
7F08            OP160   =   $F5AD   ;PRINT 'NOT FOUND'
7F08            CONTLD  =   $F41C   ;FINISH LOAD
7F08            LD16    =   $F3F8   ;LOAD FILE
7F08            RDTAPE  =   $F89A   ;TAPE READ
7F08            WRT     =   $FFD2   ;PRINT CHARACTER IN R(A)
                ;
7F08                    *=  $7F08   ; 'SYS 32520'
                ;
7F08 AE FE 7F   VICLD   LDX SART    ; CHECK TOP OF MEMORY
7F0B E4 34              CPX MEMSIZ
7F0D AD FF 7F           LDA SART+1
7F10 E5 35              SBC MEMSIZ+1
7F12 B0 0B              BCS WARM
                ;
7F14 86 34              STX MEMSIZ  ; MUST LOWER TOP OF MEMORY
7F16 86 30              STX FRETOP
7F18 AD FF 7F           LDA SART+1
7F1B 85 35              STA MEMSIZ+1
7F1D 85 31              STA FRETOP+1
                ;
7F1F A2 03      WARM    LDX #3      ; SWAP OUT/IN CHRGET ROUTIN
7F21 B5 78      LOOP    LDA CHRGOT+2,X
7F23 48                 PHA
7F24 BD FA 7F           LDA CHRGOTA-1,X
7F27 95 78              STA CHRGOT+2,X
7F29 68                 PLA
7F2A 9D FA 7F           STA CHRGOTA-1,X
7F2D CA                 DEX
7F2E D0 F1              BNE LOOP
7F30 60         STRTS   RTS
                ;
7F31 C9 3C      WEDGE   CMP #"<"    ; CHECK FOR 'UNWEDGE'
7F33 D0 08              BNE WG200
7F35 48                 PHA
7F36 A5 77              LDA TXTPTR
7F38 C9 00              CMP #0      ; FIRST CHARACTER "?"
7F3A F0 08              BEQ WCMD
                ;
7F3C 68         WG100   PLA         ; FINISH CHRGOT
7F3D C9 3A      WG200   CMP #":"
7F3F B0 EF              BCS STRTS
                ;
7F41 4C 7D 00           JMP CHRGOT+7
                ;
7F44 20 70 00 WCMD      JSR CHRGET
7F47 C9 56              CMP #"V"    ; 'V' MEANS VIC-LOAD
7F49 D0 F1              BNE WG100   ; EXIT
                ;
7F4B A2 01      WG300   LDX #1      ; SET TO CASSETTE#1
7F4D 86 D4              STX FA
7F4F CA                 DEX
7F50 86 D1              STX FNLEN
7F52 86 9D              STX VERCK
7F54 A9 02              LDA #2
7F56 85 DB              STA FNADR+1
                ;
7F58 20 70 00 WC100     JSR CHRGET  ; GET FILENAME IF PRESENT
7F5B AA                 TAX
7F5C F0 17              BEQ WC210
7F5E C9 22              CMP #QUOTE
7F60 D0 F6              BNE WC100
```

```
7F62 A6 77            LDX  TXTPTR
7F64 E8               INX
7F65 86 DA            STX  FNADR
                ;
7F67 20 70 00 WC200   JSR  CHRGET
7F6A AA               TAX
7F6B F0 08            BEQ  WC210
7F6D C9 22            CMP  #QUOTE
7F6F F0 04            BEQ  WC210
7F71 E6 D1            INC  FNLEN
7F73 D0 F2            BNE  WC200
                ;
7F75 20 95 F6 WC210   JSR  ZZZ      ; GIVE NORMAL MSGS
7F78 20 57 F8         JSR  CSTE1
7F7B 20 49 F4         JSR  LD300
7F7E A5 D1    WC215   LDA  FNLEN
7F80 F0 0B            BEQ  WC250
7F82 20 E7 7F         JSR  FVH      ; FIND 'VIC' HEADER
7F85 20 D6 F4         JSR  FAF1     ; RE-ENTER PET ROM ROUTINES
7F88 D0 08            BNE  WC270
7F8A 4C AD F5 WC220   JMP  OP160
7F8D 20 E7 7F WC250   JSR  FVH      ; FIND 'VIC' HEADER
7F90 F0 F8            BEQ  WC220
7F92 A5 96    WC270   LDA  ST
7F94 29 10            AND  #%00010000
7F96 D0 4C            BNE  WC300
                ;
7F98 E0 01            CPX  #1       ; NORMAL PROGRAM HEADER
7F9A F0 1D            BEQ  WC280
                ;
7F9C E0 03            CPX  #3       ; SPECIAL VIC 'ABSOLUTE LOAD'
7F9E D0 DE            BNE  WC215    ; FIND ANOTHER
                ;
7FA0 BD 7B 02 WC275   LDA  TAPE1+1,X  ; GET START/END
7FA3 95 FB            STA  TMP0,X   ; FROM TAPE HEADER
7FA5 CA               DEX
7FA6 10 F8            BPL  WC275
                ;
7FA8 20 17 D7         JSR  WROA     ; PRINT THEM ON SCREEN
7FAB A9 2D            LDA  #"-"
7FAD 20 D2 FF         JSR  WRT
7FB0 20 44 D7         JSR  T2T2
7FB3 20 17 D7         JSR  WROA
7FB6 B8               CLV
7FB7 50 28            BVC  WC290    ; ALWAYS
                ;
7FB9 AD 7D 02 WC280   LDA  TAPE1+3  ; GET PROGRAM LENGTH
7FBC 38               SEC
7FBD ED 7B 02         SBC  TAPE1+1
7FC0 AA               TAX
7FC1 AD 7E 02         LDA  TAPE1+4
7FC4 ED 7C 02         SBC  TAPE1+2
7FC7 A8               TAY
                ;
7FC8 A5 28            LDA  TXTTAB   ; CHANGE HEADER BYTES
7FCA 8D 7B 02         STA  TAPE1+1  ; TO START OF BASIC
7FCD A5 29            LDA  TXTTAB+1
7FCF 8D 7C 02         STA  TAPE1+2
7FD2 8A               TXA
7FD3 18               CLC           ; FIX END LOAD ADDRESS
7FD4 6D 7B 02         ADC  TAPE1+1
7FD7 8D 7D 02         STA  TAPE1+3
7FDA 98               TYA
7FDB 6D 7C 02         ADC  TAPE1+2
7FDE 8D 7E 02         STA  TAPE1+4
                ;
7FE1 20 F8 F3 WC290   JSR  LD16     ; CONTINUE THE LOAD
                ;
7FE4 4C 1C F4 WC300   JMP  CONTLD
                ;
7FE7 A5 9D    FVH     LDA  VERCK    ; FIND A 'VIC' HEADER
7FE9 48               PHA
7FEA 20 9A F8         JSR  RDTAPE
7FED A0 00            LDY  #0
7FEF B1 D6            LDA  (TBUF),Y
7FF1 C9 03            CMP  #3       ; IS IT 'ABSOLUTE' TYPE
7FF3 F0 03            BEQ  ABSLD
7FF5 4C EF F5         JMP  FAH1     ; RE-ENTER ROM ROUTINES
                ;
7FF8 4C FB F5 ABSLD   JMP  FAH2     ; GOT HEADER TYPE '03'
                ;
7FFB 4C       CHRGOTA .BYT $4C      ; SWAP INTO ZERO PAGE
7FFC 31 7F            .WOR WEDGE
                ;
7FFE 08 7F    SART    .WOR VICLD    ; PROGRAM LOCATION START
```

SUPERMON For The VIC

David A. Hook
Barrie, Ontario

Introduction

No machine language monitor is provided in the VIC ROMs. Serious users find themselves in the same position as the first generation (BASIC 1.0) of PET users. Since only a commercial product, VICMON, is available, the need for a public-domain utility seemed a worthwhile project.

Jim Butterfield has already developed TINYMON1, which may be found in Compute #20 (January 1982). This offers the equivalent to the Commodore TIM monitor.

We've already been accustomed to the benefits inherent in Supermon, Extramon and Micromon for the PET/CBM. Herewith my adaptation of Supermon for the VIC.

While we are on the subject of Jim Butterfield, I once again offer my thanks for his many contributions to the PET/CBM/VIC community. His splendid work and the donation of same to the public domain is quite remarkable. We often fail to acknowledge how rich is our store of knowledge because of this gentleman.

Features

Like its predecessors, VIC-20 SUPERMON loads and self-relocates to the top of VIC memory, regardless of the memory configuration installed. It was designed to fit in less than 2K, and I surrendered the "single-step" mode to accomplish this goal.

Make no mistake, there's a lot of time to be invested to reproduce the code of this program. Since I plan to make this available through the Toronto Pet Users Group, you may obtain this at the April meeting. It should also be part of that evening's "Copy Disk".

The syntax is identical to that of TINYMON1 and Supermon (The Transactor, Vol. II, #11 of November 1980, Compute #19 of December 1981 or The Torpet, Issue #7 of October 1981). See these references for detailed documentation.

I've copied Jim's technique of "absolute-load" files with VIC-20 SUPERMON. This means that if you SAVE memory with the monitor, the VIC won't relocate it on you when you come to re-LOAD the code.

Jim said that the PET would ignore such files, but my utility called VICLOAD covers that shortcoming.

Procedure

Apart from the investment of time, you don't really need to be familiar with machine-language to enter the program. One of Supermon's benefits is its value for beginners in ML programming.

You'll need a PET/CBM with Upgrade (BASIC 2.0) or BASIC 4.0 ROM installed. Start with a freshly powered-up machine. Enter the ML monitor by typing 'SYS4' and hitting 'RETURN'.

There are 20 screens of information to be entered before you're done. Each requires the same procedure. Only the first (line entry) is slightly different:

1. Immediately after the ".", type the range of memory to be displayed, like so:

.M 0028 0028 <RETURN>

One line of hexadecimal digits is shown on the screen.

2. Move the cursor back to this line and type over the digits with the values:

.: 0028 01 04 DF 0D DF 0D DF 0D <RETURN>

Don't forget to hit the 'RETURN' key at the end of each line, or else the new values won't be remembered.

Now type in the starting and ending addresses for each of the 20 screens to be changed. For example, the first would be:

.M 0400 0478 <RETURN>

Copy the values shown in the accompanying "hex dump" for each block shown. Type right over the existing values, (which will probably show as "AA" on your screen). Do your best to double-check before proceding with the next block.

Repeat until the block from $0D80 to $0DF8 is done. Now exit the monitor, with:

.X <RETURN>

NOW SAVE AND VERIFY THE PROGRAM. Do not pass "GO", do not collect $200... do this first!

Checking

Because of the 2000+ entries you've made, the chance for error is high. Several "Immediate-" or "Calculator- Mode" statements are provided to verify your work.

These statements do a "check-sum" on the total program, each multiple of 5-screens (4 check-sums), and each "line" of 8-entries.

Enter the statements shown, with no line numbers please! On hitting 'RETURN', the total will be shown immediately below.

If your first one shows "283370", then it's probably perfect. Go to the head of the class, and start using VIC-20 SUPERMON.

Otherwise, work your way though the next four, noting which are correct. Each of these totals are a composite of 80 lines of entry, (or 5-screens, as we put them in).

For any incorrect block from above, there is a corresponding statement to type. This will give the individual totals for each line of entry. Mark the lines that are different. You will now have to re-enter the ML monitor and make the necessary corrections. ReSAVE this version and re-do the checksum until it's correct.

Operation

Disconnect your cassette recorder (power off the PET first, please). Reconnect to the VIC, turn everything on, LOAD VIC-20 SUPERMON and RUN it.

You should be greeted with the so-called "Register Display", as on the PET/CBM. Unless it's a B & W display, you'll see several usages of VIC colour.

Any entries you make will be in blue, while the VIC variously displays red for register headings or errors, purple for normal addresses, black for memory bytes and green for disassembly mnemonics or "next" addresses when assembling code.

Since we are dealing with a 22-character wide screen, the disassembly consumes two lines. The mnemonics are pushed to the right on the second line. More locations would have fit, but I chose to keep the actual "bytes" display. This of course permits you to change them, and causes an automatic redisplay of the same range of addresses.

The goal of "under 2K" was met (by 3 bytes), but the single-step had to be sacrificed. Someone else may re-work Micromon for VIC, but its 4K size is of no use to the "3583 BYTES FREE" crowd.

In Closing...

If I use this half as much as Supermon for PET, it will have been worth the adaptation effort. Without Supermon 1.0 (for Original ROM), this would not have been feasible. A deep bow towards 14 Brooklyn Avenue, Toronto.

David A. Hook
58 Steel Street
BARRIE, Ontario
L4M 2E9
(705) 726-8126

Editor's Note

VICMON is available now from Commodore dealers. Unlike Supermon for the VIC, VICMON comes on a plug-in cartridge. VICMON is a 4K program and doesn't use any of the VICs RAM, leaving more room for your programs. Several additional features include scrolling backwards and forwards, quicktrace, single step, and more! If Daves' Supermon for the VIC wets your appetite for further exploration into the realm of machine language, you'll find VICMON a welcome addition to your programming toolbox!

```
.
.: 0400 00 1A 04 64 00 99 22 93
.: 0408 12 1D 1D 1D 1D 53 55 50
.: 0410 45 52 20 56 49 43 4D 4F
.: 0418 4E 00 2F 04 6E 00 99 22
.: 0420 11 44 41 56 49 44 20 41
.: 0428 2E 20 48 4F 4F 4B 00 44
.: 0430 04 70 00 99 22 11 46 52
.: 0438 4F 4D 20 53 55 50 45 52
.: 0440 4D 4F 4E 00 5E 04 73 00
.: 0448 99 22 11 42 59 20 4A 49
.: 0450 4D 20 42 55 54 54 45 52
.: 0458 46 49 45 4C 44 00 79 04
.: 0460 78 00 9E 28 C2 28 34 33
.: 0468 29 AA 32 35 36 AC C2 28
.: 0470 34 34 29 AA 31 32 37 29
.: 0478 00 00 00 AA AA AA AA AA
.
.
.: 0480 A5 2D 85 22 A5 2E 85 23
.: 0488 A5 37 85 24 A5 38 85 25
.: 0490 A0 00 A5 22 D0 02 C6 23
.: 0498 C6 22 B1 22 D0 3C A5 22
.: 04A0 D0 02 C6 23 C6 22 B1 22
.: 04A8 F0 21 85 26 A5 22 D0 02
.: 04B0 C6 23 C6 22 B1 22 18 65
.: 04B8 24 AA A5 26 65 25 48 A5
.: 04C0 37 D0 02 C6 38 C6 37 68
.: 04C8 91 37 8A 48 A5 37 D0 02
.: 04D0 C6 38 C6 37 68 91 37 18
.: 04D8 90 B6 C9 BF D0 ED A5 37
.: 04E0 85 33 A5 38 85 34 6C 37
.: 04E8 00 AA AA AA AA AA AA AA
.: 04F0 BF 78 AD E8 FF 00 8D 16
.: 04F8 03 AD E9 FF 00 8D 17 03
.
.
.: 0500 A9 80 20 90 FF 58 00 00
.: 0508 D8 68 85 05 68 85 04 68
.: 0510 85 03 68 85 02 68 AA 68
.: 0518 A8 38 8A E9 02 85 01 98
.: 0520 E9 00 00 85 00 00 BA 86
.: 0528 06 20 CE F8 00 A2 42 A9
.: 0530 2A 20 43 FA 00 A9 52 D0
.: 0538 1B A9 1F 20 D2 FF A9 00
.: 0540 00 85 26 A2 0D A9 2E 20
.: 0548 43 FA 00 20 BB FA 00 C9
.: 0550 2E F0 F9 C9 20 F0 F5 A2
.: 0558 0E DD BB FF 00 D0 0C 8A
.: 0560 0A AA BD CB FF 00 48 BD
.: 0568 CA FF 00 48 60 CA 10 EC
.: 0570 4C FC FA 00 A5 C1 85 01
.: 0578 A5 C2 85 00 00 60 A9 05
.
.
.: 0580 85 1D A9 90 20 D2 FF A0
.: 0588 00 00 20 CB F8 00 B1 C1
.: 0590 20 34 FA 00 20 D3 F8 00
.: 0598 C6 1D D0 F1 60 20 8B FA
.: 05A0 00 90 0B A2 00 00 81 C1
.: 05A8 C1 C1 F0 03 4C FC FA 00
.: 05B0 20 D3 F8 00 C6 1D 60 A9
.: 05B8 02 85 C1 A9 00 00 85 C2
.: 05C0 60 98 48 20 CE F8 00 A9
.: 05C8 9C 20 D2 FF 68 A2 2E 4C
.: 05D0 43 FA 00 A9 20 2C A9 0D
.: 05D8 4C D2 FF E6 C1 D0 06 E6
.: 05E0 C2 D0 02 E6 26 60 A9 1C
.: 05E8 20 D2 FF A2 00 00 BD EA
.: 05F0 FF 00 20 D2 FF E8 E0 16
.: 05F8 D0 F5 A0 3B 20 BB F8 00
.
.
.: 0600 A5 00 00 20 34 FA 00 A5
.: 0608 01 20 34 FA 00 20 B2 F8
.: 0610 00 20 81 F8 00 F0 57 20
.: 0618 BB FA 00 20 7C FA 00 90
.: 0620 2E 20 6C FA 00 20 BB FA
.: 0628 00 20 7C FA 00 90 23 20
.: 0630 6C FA 00 20 70 F7 F0 3C
.: 0638 A6 26 D0 38 A5 C3 C5 C1
.: 0640 A5 C4 E5 C2 90 2E A0 3A
.: 0648 20 BB F8 00 20 2D FA 00
.: 0650 20 81 F8 00 F0 E0 4C FC
.: 0658 FA 00 20 7C FA 00 90 03
.: 0660 20 78 F8 00 20 B2 F8 00
.: 0668 F0 05 20 7C FA 00 90 EB
.: 0670 A9 05 85 1D 20 BB FA 00
.: 0678 20 9C F8 00 D0 F8 4C 44
.
.
.: 0680 F8 00 20 CF FF C9 0D F0
.: 0688 0C C9 20 D0 D1 20 7C FA
.: 0690 00 90 03 20 78 F8 00 A6
.: 0698 06 9A 78 A5 00 00 48 A5
.: 06A0 01 48 A5 02 48 A5 03 A6
.: 06A8 04 A4 05 40 78 A6 06 9A
.: 06B0 6C 02 C0 A0 01 84 BA 84
.: 06B8 B9 88 84 B7 84 90 84 93
.: 06C0 A9 40 85 BB A9 02 85 BC
.: 06C8 20 CF FF C9 20 F0 F9 C9
.: 06D0 0D F0 37 C9 22 D0 34 20
.: 06D8 CF FF C9 22 F0 0F C9 0D
.: 06E0 F0 28 91 BB E6 B7 C8 C0
.: 06E8 10 F0 20 D0 EA 20 CF FF
.: 06F0 C9 0D F0 16 C9 2C D0 DD
.: 06F8 20 8B FA 00 29 0F F0 EA
.
```

```
.: 0700 C9 03 F0 E6 85 BA 20 CF
.: 0708 FF C9 0D 60 4C FC FA 00
.: 0710 20 8F F9 00 D0 F8 20 4F
.: 0718 F5 20 4C FA 00 A5 90 29
.: 0720 10 D0 EC 4C 44 F8 00 20
.: 0728 8F F9 00 C9 2C D0 E2 20
.: 0730 7C FA 00 20 6C FA 00 20
.: 0738 CF FF C9 2C D0 D5 20 7C
.: 0740 FA 00 A5 C1 85 AE A5 C2
.: 0748 85 AF 20 6C FA 00 20 CF
.: 0750 FF C9 0D D0 C0 20 82 F6
.: 0758 4C 44 F8 00 A5 C2 20 34
.: 0760 FA 00 A5 C1 48 4A 4A 4A
.: 0768 4A 20 63 FA 00 AA 68 29
.: 0770 0F 20 63 FA 00 48 8A 20
.: 0778 D2 FF 68 4C D2 FF 20 55
.
.: 0780 FA 00 2C 2D 91 30 F8 60
.: 0788 20 70 F7 D0 08 A9 03 85
.: 0790 9A A9 00 00 85 99 60 09
.: 0798 30 C9 3A 90 02 69 06 60
.: 07A0 A2 02 B5 C0 48 B5 C2 95
.: 07A8 C0 68 95 C2 CA D0 F3 60
.: 07B0 20 8B FA 00 90 02 85 C2
.: 07B8 20 8B FA 00 90 02 85 C1
.: 07C0 60 A9 00 00 85 2A 20 BB
.: 07C8 FA 00 C9 20 D0 09 20 BB
.: 07D0 FA 00 C9 20 D0 0E 18 60
.: 07D8 20 B2 FA 00 0A 0A 0A 0A
.: 07E0 85 2A 20 BB FA 00 20 B2
.: 07E8 FA 00 05 2A 38 60 C9 3A
.: 07F0 90 02 69 08 29 0F 60 20
.: 07F8 CF FF C9 0D D0 F8 68 68
.
.: 0800 4C 44 F8 00 A2 02 2C A2
.: 0808 00 00 B4 C1 D0 08 B4 C2
.: 0810 D0 02 E6 26 D6 C2 D6 C1
.: 0818 60 20 BB FA 00 C9 20 F0
.: 0820 F9 60 A9 00 00 8D 00 00
.: 0828 01 20 DB FA 00 20 92 FA
.: 0830 00 20 7F FA 00 90 09 60
.: 0838 20 BB FA 00 20 7C FA 00
.: 0840 B0 DE A6 06 9A A9 1C 20
.: 0848 D2 FF A9 3F 20 D2 FF 4C
.: 0850 44 F8 00 20 CB F8 00 CA
.: 0858 D0 FA 60 E6 C3 D0 02 E6
.: 0860 C4 60 A2 02 B5 C0 48 B5
.: 0868 27 95 C0 68 95 27 CA D0
.: 0870 F3 60 A5 28 A4 29 4C 35
.: 0878 FB 00 A5 C3 A4 C4 38 E5
.
.: 0880 C1 85 1E 98 E5 C2 A8 05
.: 0888 1E 60 20 E3 FA 00 20 6C
.: 0890 FA 00 20 F4 FA 00 20 1A
.: 0898 FB 00 20 F4 FA 00 20 31
.: 08A0 FB 00 20 6C FA 00 90 15
.: 08A8 A6 26 D0 64 20 2A FB 00
.: 08B0 90 5F A1 C1 81 C3 20 13
.: 08B8 FB 00 20 D3 F8 00 D0 EB
.: 08C0 20 2A FB 00 18 A5 1E 65
.: 08C8 C3 85 C3 98 65 C4 85 C4
.: 08D0 20 1A FB 00 A6 26 D0 3D
.: 08D8 A1 C1 81 C3 20 2A FB 00
.: 08E0 B0 34 20 C7 FA 00 20 CA
.: 08E8 FA 00 4C 7F FB 00 20 E3
.: 08F0 FA 00 20 6C FA 00 20 F4
.: 08F8 FA 00 20 6C FA 00 20 BB
.
.: 0900 FA 00 20 8B FA 00 90 14
.: 0908 85 1D A6 26 D0 11 20 31
.: 0910 FB 00 90 0C A5 1D 81 C1
.: 0918 20 D3 F8 00 D0 EE 4C FC
.: 0920 FA 00 4C 44 F8 00 20 E3
.: 0928 FA 00 20 6C FA 00 20 F4
.: 0930 FA 00 20 6C FA 00 20 BB
.: 0938 FA 00 A2 00 00 20 BB FA
.: 0940 00 C9 27 D0 14 20 BB FA
.: 0948 00 9D 10 02 E8 20 CF FF
.: 0950 C9 0D F0 22 E0 20 D0 F1
.: 0958 F0 1C 8E 00 00 01 20 92
.: 0960 FA 00 90 C6 9D 10 02 E8
.: 0968 20 CF FF C9 0D F0 09 20
.: 0970 8B FA 00 90 B6 E0 20 D0
.: 0978 EC 86 1C 20 CE F8 00 A2
.
.: 0980 00 00 A0 00 00 B1 C1 DD
.: 0988 10 02 D0 0C C8 E8 E4 1C
.: 0990 D0 F3 20 2D FA 00 20 CB
.: 0998 F8 00 20 D3 F8 00 A6 26
.: 09A0 D0 92 20 31 FB 00 B0 DD
.: 09A8 4C 44 F8 00 20 E3 FA 00
.: 09B0 85 20 A5 C2 85 21 A9 08
.: 09B8 A2 00 00 85 27 86 28 A9
.: 09C0 93 20 D2 FF A9 0B 85 1D
.: 09C8 20 67 FC 00 20 D1 FC 00
.: 09D0 85 C1 84 C2 C6 1D D0 F2
.: 09D8 A9 91 AA 20 43 FA 00 4C
.: 09E0 44 F8 00 A0 2C 20 BB F8
.: 09E8 00 20 CB F8 00 20 2D FA
.: 09F0 00 20 CB F8 00 A2 00 00
.: 09F8 A1 C1 20 E0 FC 00 48 A9
.
```

```
.:  0A00 90 20 D2 FF 20 26 FD 00
.:  0A08 A9 1E 20 D2 FF 68 20 3C
.:  0A10 FD 00 A2 06 E0 03 D0 12
.:  0A18 A4 1F F0 0E A5 2A C9 E8
.:  0A20 B1 C1 B0 1C 20 C9 FC 00
.:  0A28 88 D0 F2 06 2A 90 0E BD
.:  0A30 2E FF 00 20 A6 FD 00 BD
.:  0A38 34 FF 00 F0 03 20 A6 FD
.:  0A40 00 CA D0 D5 60 20 D4 FC
.:  0A48 00 AA E8 D0 01 C8 98 20
.:  0A50 C9 FC 00 8A 86 1C 20 34
.:  0A58 FA 00 A6 1C 60 A5 1F 38
.:  0A60 A4 C2 AA 10 01 88 65 C1
.:  0A68 90 01 C8 60 A8 4A 90 0B
.:  0A70 4A B0 17 C9 22 F0 13 29
.:  0A78 07 09 80 4A AA BD DD FE
.

.:  0A80 00 B0 04 4A 4A 4A 4A 29
.:  0A88 0F D0 04 A0 80 A9 00 00
.:  0A90 AA BD 21 FF 00 85 2A 29
.:  0A98 03 85 1F 98 29 8F AA 98
.:  0AA0 A0 03 E0 8A F0 0B 4A 90
.:  0AA8 08 4A 4A 09 20 88 D0 FA
.:  0AB0 C8 88 D0 F2 60 B1 C1 20
.:  0AB8 C9 FC 00 A2 01 20 0C FB
.:  0AC0 00 C4 1F C8 90 F1 A2 03
.:  0AC8 C4 27 90 F2 60 A8 B9 3B
.:  0AD0 FF 00 85 28 B9 7B FF 00
.:  0AD8 85 29 A9 00 00 A0 05 06
.:  0AE0 29 26 28 2A 88 D0 F8 69
.:  0AE8 3F 20 D2 FF CA D0 EC 4C
.:  0AF0 CB F8 00 20 E3 FA 00 20
.:  0AF8 6C FA 00 20 F4 FA 00 20
.

.:  0B00 6C FA 00 A9 09 A2 00 00
.:  0B08 85 27 86 28 20 CE F8 00
.:  0B10 20 6F FC 00 20 D1 FC 00
.:  0B18 85 C1 84 C2 20 70 F7 F0
.:  0B20 05 20 31 FB 00 B0 E9 4C
.:  0B28 44 F8 00 20 E3 FA 00 A9
.:  0B30 03 85 1D 20 BB FA 00 20
.:  0B38 9C F8 00 D0 F8 A5 20 85
.:  0B40 C1 A5 21 85 C2 4C 47 FC
.:  0B48 00 C5 28 F0 03 20 D2 FF
.:  0B50 60 20 E3 FA 00 20 6C FA
.:  0B58 00 8E 11 02 A2 03 20 DB
.:  0B60 FA 00 48 CA D0 F9 A2 03
.:  0B68 68 38 E9 3F A0 05 4A 6E
.:  0B70 11 02 6E 10 02 88 D0 F6
.:  0B78 CA D0 ED A2 02 20 CF FF
.
```

```
.:  0B80 C9 0D F0 1E C9 20 F0 F5
.:  0B88 20 D4 FE 00 B0 0F 20 9F
.:  0B90 FA 00 A4 C1 84 C2 85 C1
.:  0B98 A9 30 9D 10 02 E8 9D 10
.:  0BA0 02 E8 D0 DB 86 28 A2 00
.:  0BA8 00 86 26 F0 04 E6 26 F0
.:  0BB0 75 A2 00 00 86 1D A5 26
.:  0BB8 20 E0 FC 00 A6 2A 86 29
.:  0BC0 AA BC 3B FF 00 BD 7B FF
.:  0BC8 00 20 BD FE 00 D0 E3 A2
.:  0BD0 06 E0 03 D0 19 A4 1F F0
.:  0BD8 15 A5 2A C9 E8 A9 30 B0
.:  0BE0 21 20 C3 FE 00 D0 CC 20
.:  0BE8 C5 FE 00 D0 C7 88 D0 EB
.:  0BF0 06 2A 90 0B BC 34 FF 00
.:  0BF8 BD 2E FF 00 20 BD FE 00
.

.:  0C00 D0 B5 CA D0 D1 F0 0A 20
.:  0C08 BC FE 00 D0 AB 20 BC FE
.:  0C10 00 D0 A6 A5 28 C5 1D D0
.:  0C18 A0 20 6C FA 00 A4 1F F0
.:  0C20 2B A5 29 C9 9D D0 1D 20
.:  0C28 31 FB 00 90 0A 98 D0 04
.:  0C30 A6 1E 10 0A 4C FC FA 00
.:  0C38 C8 D0 FA A6 1E 10 F6 CA
.:  0C40 CA 8A A4 1F D0 03 B9 C2
.:  0C48 00 00 91 C1 88 D0 F8 A5
.:  0C50 26 91 C1 20 D1 FC 00 85
.:  0C58 C1 84 C2 A0 41 20 BB F8
.:  0C60 00 20 CB F8 00 A9 1E 20
.:  0C68 D2 FF 20 2D FA 00 20 CB
.:  0C70 F8 00 A9 1F 20 D2 FF 4C
.:  0C78 B1 FD 00 A8 20 C3 FE 00
.

.:  0C80 D0 11 98 F0 0E 86 1C A6
.:  0C88 1D DD 10 02 08 E8 86 1D
.:  0C90 A6 1C 28 60 C9 30 90 03
.:  0C98 C9 47 60 38 60 40 02 45
.:  0CA0 03 D0 08 40 09 30 22 45
.:  0CA8 33 D0 08 40 09 40 02 45
.:  0CB0 33 D0 08 40 09 40 02 45
.:  0CB8 B3 D0 08 40 09 00 00 22
.:  0CC0 44 33 D0 8C 44 00 00 11
.:  0CC8 22 44 33 D0 8C 44 9A 10
.:  0CD0 22 44 33 D0 08 40 09 10
.:  0CD8 22 44 33 D0 08 40 09 62
.:  0CE0 13 78 A9 00 00 21 81 82
.:  0CE8 00 00 00 00 59 4D 91 92
.:  0CF0 86 4A 85 9D 2C 29 2C 23
.:  0CF8 28 24 59 00 00 58 24 24
.
```

```
.:  0D00 00 00 1C 8A 1C 23 5D 8B
.:  0D08 1B A1 9D 8A 1D 23 9D 8B
.:  0D10 1D A1 00 00 29 19 AE 69
.:  0D18 A8 19 23 24 53 1B 23 24
.:  0D20 53 19 A1 00 00 1A 5B 5B
.:  0D28 A5 69 24 24 AE AE A8 AD
.:  0D30 29 00 00 7C 00 00 15 9C
.:  0D38 6D 9C A5 69 29 53 84 13
.:  0D40 34 11 A5 69 23 A0 D8 62
.:  0D48 5A 48 26 62 94 88 54 44
.:  0D50 C8 54 68 44 E8 94 00 00
.:  0D58 B4 08 84 74 B4 28 6E 74
.:  0D60 F4 CC 4A 72 F2 A4 8A 00
.:  0D68 00 AA A2 A2 74 74 74 72
.:  0D70 44 68 B2 32 B2 00 00 22
.:  0D78 00 00 1A 1A 26 26 72 72
.
.
.:  0D80 88 C8 C4 CA 26 48 44 44
.:  0D88 A2 C8 3A 3B 52 4D 47 58
.:  0D90 4C 53 54 46 48 44 50 2C
.:  0D98 41 4C F9 00 3F F9 00 DD
.:  0DA0 F8 00 06 F9 00 60 F9 00
.:  0DA8 87 F9 00 E9 F9 00 FD F9
.:  0DB0 00 40 FB 00 94 FB 00 C2
.:  0DB8 FB 00 35 FC 00 5D FD 00
.:  0DC0 8B FD 00 AD FD 00 17 F8
.:  0DC8 00 0D 20 20 20 50 43 20
.:  0DD0 20 53 52 20 41 43 20 58
.:  0DD8 52 20 59 52 20 53 50 AA
.:  0DE0 AA AA AA AA AA AA AA AA
.:  0DE8 AA AA AA AA AA AA AA AA
.:  0DF0 AA AA AA AA AA AA AA AA
.:  0DF8 AA AA AA AA AA AA AA AA
.
```

```
T=0:FORJ=1024TO3550:T=T+PEEK(J):NEXT:?T
 283370

READY.
T=0:FORJ=1024TO1663:T=T+PEEK(J):NEXT:?T
 68631

READY.
T=0:FORJ=1664TO2303:T=T+PEEK(J):NEXT:?T
 77155

READY.
T=0:FORJ=2304TO2943:T=T+PEEK(J):NEXT:?T
 74768

READY.
T=0:FORJ=2944TO3550:T=T+PEEK(J):NEXT:?T
 62816

READY.
```

```
)RJ=1024TO1663STEP8:T=0:FORK=JTOJ+7:T=T
>EEK(K):NEXT:?T,:NEXT
464          382          565          426
474          451          472          587
447          538          579          481
555          774          510          850
756          780          802          910
386          853          801          784
776          840          835          1383
453          1190         1134         831
316          803          753          883
586          889          850          893
593          987          1415         1035
1088         1079         1070         762
1132         853          825          1193
539          1207         983          824
975          1041         744          1408
965          1082         1230         1139
564          793          768          987
905          617          1049         1218
1192         794          1201         803
1358         1030         805          1036
```

```
FORJ=2304TO2943STEP8:T=0:FORK=JTOJ+7:T=T
+PEEK(K):NEXT:?T,:NEXT
 835          672          923          1265
 901          916          859          881
 937          901          1193         589
 999          989          1179         1046
 751          926          1013         943
 1083         901          867          677
 986          880          1329         909
 987          810          645          1103
 964          892          874          1089
 1059         981          941          1001
 1215         995          837          792
 975          838          808          1052
 517          684          863          825
 994          791          1284         911
 977          1129         991          514
 858          1282         992          916
 698          832          888          1283
 822          994          666          1190
 1117         977          995          577
 1146         805          737          1305
```

```
)RJ=1664TO2303STEP8:T=0:FORK=JTOJ+7:T=T
?EEK(K):NEXT:?T,:NEXT
1196         1068         713          682
1546         683          913          1191
1045         1417         835          1166
1417         1224         1150         951
1232         1143         991          953
1384         1103         796          1284
1274         937          1277         835
902          770          638          1227
1376         912          714          660
1133         1388         894          893
959          919          825          500
1254         708          443          1340
1762         963          1293         1038
1555         930          658          875
953          1270         1001         1419
1082         1082         878          1256
1104         775          834          858
1306         837          968          1185
1545         1301         782          1003
943          963          916          859
```

```
FORJ=2944TO3551STEP8:T=0:FORK=JTOJ+7:T=T
+PEEK(K):NEXT:?T,:NEXT
 1202         880          1259         797
 997          924          645          891
 1239         1072         901          1054
 958          1437         698          965
 1290         1295         1013         985
 876          818          800          1318
 1125         1095         1002         1211
 714          1027         1021         1079
 959          671          726          655
 443          475          475          502
 552          739          458          540
 600          457          662          325
 461          843          535          445
 477          1031         342          810
 848          734          836          882
 1180         956          612          356
 980          797          577          923
 848          1368         908          902
 1089         288          481          564
```

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
0	00		@	end-line	BRK	0
1	01		A		ORA(I,X)	1
2	02		B			2
3	03		C			3
4	04		D			4
5	05	white	E		ORA Z	5
6	06		F		ASL Z	6
7	07	bell	G			7
8	08	lock	H		PHP	8
9	09	unlock	I		ORA #	9
10	0A		J		ASL A	10
11	0B		K			11
12	0C		L			12
13	0D	car ret	M		ORA	13
14	0E	text	N		ASL	14
15	0F	top-left	O			15
16	10		P		BPL	16
17	11	cur down	Q		ORA(I);Y	17
18	12	reverse	R			18
19	13	cur home	S			19
20	14	delete	T			20
21	15	del.line	U		ORA Z,X	21
22	16	er.begin	V		ASL Z,X	22
23	17		W			23
24	18		X		CLC	24
25	19	scr.up	Y		ORA Y	25
26	1A		Z			26
27	1B		[			27
28	1C	red	\			28
29	1D	cur right	]		ORA X	29
30	1E	green	↑		ASL X	30
31	1F	blue	←			31
32	20	space	space	space	JSR	32
33	21	!	!	!	AND(I,X)	33
34	22	"	"	"		34
35	23	#	#	#		35
36	24	$	$	$	BIT Z	36
37	25	%	%	%	AND Z	37
38	26	&	&	&	ROL Z	38
39	27	'	'	'		39
40	28	(	(	(	PLP	40
41	29	)	)	)	AND #	41
42	2A	*	*	*	ROL A	42
43	2B	+	+	+		43
44	2C	,	,	,	BIT	44
45	2D	-	-	-	AND	45
46	2E	.	.	.	ROL	46
47	2F	/	/	/		47
48	30	0	0	0	BMI	48
49	31	1	1	1	AND(I),Y	49
50	32	2	2	2		50
51	33	3	3	3		51
52	34	4	4	4		52
53	35	5	5	5	AND Z,X	53
54	36	6	6	6	ROL Z,X	54
55	37	7	7	7		55
56	38	8	8	8	SEC	56
57	39	9	9	9	AND Y	57
58	3A	:	:	:	CLI	58
59	3B	;	;	;		59
60	3C	<	<	<		60
61	3D	=	=	=	AND X	61
62	3E	>	>	>	ROL X	62
63	3F	?	?	?		63

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
64	40	@		@	RTI	64
65	41	A		A	EOR(I,X)	65
66	42	B		B		66
67	43	C		C		67
68	44	D		D		68
69	45	E		E	EOR Z	69
70	46	F		F	LSR Z	70
71	47	G		G		71
72	48	H		H	PHA	72
73	49	I		I	EOR #	73
74	4A	J		J	LSR A	74
75	4B	K		K		75
76	4C	L		L	JMP	76
77	4D	M		M	EOR	77
78	4E	N		N	LSR	78
79	4F	O		O		79
80	50	P		P	BVC	80
81	51	Q		Q	EOR(I),Y	81
82	52	R		R		82
83	53	S		S		83
84	54	T		T		84
85	55	U		U	EOR Z,X	85
86	56	V		V	LSR Z,X	86
87	57	W		W		87
88	58	X		X	CLI	88
89	59	Y		Y	EOR Y	89
90	5A	Z		Z		90
91	5B	[		[		91
92	5C	\		\		92
93	5D	]		]	EOR X	93
94	5E	↑		↑	LSR X	94
95	5F	←		←		95
96	60				RTS	96
97	61				ADC(I,X)	97
98	62					98
99	63					99
100	64					100
101	65				ADC Z	101
102	66				ROR Z	102
103	67					103
104	68				PLA	104
105	69				ADC #	105
106	6A				ROR A	106
107	6B					107
108	6C				JMP(I)	108
109	6D				ADC	109
110	6E				ROR	110
111	6F					111
112	70				BVS	112
113	71				ADC(I),Y	113
114	72					114
115	73					115
116	74					116
117	75				ADC Z,X	117
118	76				ROR Z,X	118
119	77					119
120	78				SEI	120
121	79				ADC Y	121
122	7A					122
123	7B					123
124	7C					124
125	7D				ADC X	125
126	7E				ROR X	126
127	7F					127

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
128	80			END		128
129	81			FOR	STA(I,X)	129
130	82			NEXT		130
131	83			DATA		131
132	84			INPUT#	STY Z	132
133	85	f1		INPUT	STA Z	133
134	86	f3		DIM	STX Z	134
135	87	f5		READ		135
136	88	f7		LET	DEY	136
137	89	f2		GOTO		137
138	8A	f4		RUN	TXA	138
139	8B	f6		IF		139
140	8C	f8		RESTORE	STY	140
141	8D	car ret		GOSUB	STA	141
142	8E	graphic		RETURN	STX	142
143	8F	bta.rt		REM		143
144	90	black		STOP	BCC	144
145	91	cur up		ON	STA(I),Y	145
146	92	rvs off		WAIT		146
147	93	clear		LOAD		147
148	94	invert		SAVE	STY Z,X	148
149	95	ins.line		VERIFY	STA Z,X	149
150	96	er.end		DEF	STX Z,Y	150
151	97			POKE		151
152	98			PRINT#	TYA	152
153	99	scr.dn		PRINT	STA Y	153
154	9A			CONT	TXS	154
155	9B			LIST		155
156	9C	magenta		CLR		156
157	9D	cur left		CMD	STA X	157
158	9E	yellow		SYS		158
159	9F	cyan		OPEN		159
160	A0			CLOSE	LDY #	160
161	A1			GET	LDA(I,X)	161
162	A2			NEW	LDX #	162
163	A3			TAB(		163
164	A4			TO	LDY Z	164
165	A5			FN	LDA Z	165
166	A6			SPC(	LDX Z	166
167	A7			THEN		167
168	A8			NOT	TAY	168
169	A9			STEP	LDA #	169
170	AA			+	TAX	170
171	AB			-		171
172	AC			*	LDY	172
173	AD			/	LDA	173
174	AE			↑	LDX	174
175	AF			AND		175
176	B0			OR	BCS	176
177	B1			>	LDA(I),Y	177
178	B2			=		178
179	B3			<		179
180	B4			SGN	LDY Z,X	180
181	B5			INT	LDA Z,X	181
182	B6			ABS	LDX Z,Y	182
183	B7			USR		183
184	B8			FRE	CLV	184
185	B9			POS	LDA Y	185
186	BA			SQR	TSX	186
187	BB			RND		187
188	BC			LOG	LDY X	188
189	BD			EXP	LDA X	189
190	BE			COS	LDX Y	190
191	BF			SIN		191

DECIMAL	HEX	ASCII	SCREEN	BASIC	6502	DECIMAL
192	C0			TAN	CPY #	192
193	C1			ATN	CMP(I),X	193
194	C2			PEEK		194
195	C3			LEN		195
196	C4			STR$	CPY Z	196
197	C5			VAL	CMP Z	197
198	C6			ASC	DEC Z	198
199	C7			CHR$		199
200	C8			LEFT$	INY	200
201	C9			RIGHT$	CMP #	201
202	CA			MID$	DEX	202
203	CB			GO		203
204	CC			CONCAT	CPY	204
205	CD			DOPEN	CMP	205
206	CE			DCLOSE	DEC	206
207	CF			RECORD		207
208	D0			HEADER	BNE	208
209	D1			COLLECT	CMP(I),Y	209
210	D2			BACKUP		210
211	D3			COPY		211
212	D4			APPEND		212
213	D5			DSAVE	CMP Z,X	213
214	D6			DLOAD	DEC Z,X	214
215	D7			CATALOG		215
216	D8			RENAME	CLD	216
217	D9			SCRATCH	CMP Y	217
218	DA			DIRECTORY		218
219	DB					219
220	DC					220
221	DD				CMP X	221
222	DE				DEC X	222
223	DF					223
224	E0				CPX #	224
225	E1				SBC(I,X	225
226	E2					226
227	E3					227
228	E4				CPX Z	228
229	E5				SBC Z	229
230	E6				INC Z	230
231	E7					231
232	E8				INX	232
233	E9				SBC #	233
234	EA				NOP	234
235	EB					235
236	EC				CPX	236
237	ED				SBC	237
238	EE				INC	238
239	EF					239
240	F0				BEQ	240
241	F1				SBC(I),Y	241
242	F2					242
243	F3					243
244	F4					244
245	F5				SBC Z,X	245
246	F6				INC Z,X	246
247	F7					247
248	F8				SED	248
249	F9				SBC Y	249
250	FA					250
251	FB					251
252	FC					252
253	FD				SBC X	253
254	FE				INC X	254
255	FF					255

VIC 6560 Chip

$9000	Interlace	Left Margin (÷4)
$9001	Top Margin (÷25)	
$9002	Screen Ad bit 9	# Columns (÷22)
$9003	bit 0	# Rows (÷23) / Double Char
$9004	Input Raster Value: bits 8-1	
$9005	Screen Address bits 13-10	Character Address bits 13-10
$9006	Light Pen Input	Horizontal
$9007		Vertical
$9008	Paddle Inputs	X
$9009		Y
$900A	ON	Voice 1
$900B	ON	Voice 2
$900C	ON	Voice 3
$900D	ON	Noise
$900E	Multi-Colour Mode (=0)	Sound Amplitude
$900F	Screen Background Colour	Foregnd /Backg / Frame Colour

36864	
36865	
36866	
36867	
36868	
36869	
36870	
36871	
36872	
36873	
36874	
36875	
36876	
36877	
36878	
36879	

VIC 6522 Usage

$9110	DSR In / CTS In / DCD In / RI In / DTR out / RTS out / Data In	RS-232 Interface or, Parallel User Port
$9111	Unused - see $911F	
$9112	DDRB (for $9110)	
$9113	DDRA (for $911F)	
$9114	T1-L	RS232 Send Speed;
$9115	T1-H	Tape Write timing
$9116	T1 Latch	
$9117	T1 Latch H	
$9118	T2-L	RS-232 Input timing
$9119	T2-H	
$911A	Shift Register (*unused)	
$911B	T1 Control / T2 Ctl / Shift Reg Control / PB LE / PA LE	
$911C	CB2: RS232 Send / CB1 C / CA2: Tape motor ctrl / CA1 Ctl	
$911D	DAV: / T1 / T2 / CB1: RS232 In / CB1: RESTORE button	
$911E	ATN out / Tape sense / Joysticks: Button / Left / Down / Up / -Serial Data In / -Serial Clk In	
$911F		

Jim Butterfield

37136	
37137	
37138	
37139	
37140	
37141	
37142	
37143	
37144	
37145	
37146	
37147	
37148	
37149	
37150	
37151	

VIC 6522 Usage

$9120	Joystk Right / Tape Out	Keyboard Row Select
$9121	Keyboard Column Input	
$9122	DDRB (for $9120)	
$9123	DDRA (for $9121)	
$9124	T1-L	Cassette Tape Read;
$9125	T1-H	Keyboard & Clock
$9126	T1 Latch	Interrupt timing
$9127	T1-H Latch	
$9128	T2-L	Serial Bus Timing
$9129	T2-H	Tape R/W Timing
$912A	Shift Register (*Unused)	
$912B	T1 Control / T2 Ctrl / Shift Register Control / PB LE / PA LE	
$912C	Serial Bus Data Out / CB1 Cntl / Serial Clock Line out / CA1 Cntl	
$912D	DAV: / T1 / T2 / CB1: SRQ In / CB1: Tape In	
$912E	*Unused: see $9121	
$912F		

37152	
37153	
37154	
37155	
37156	
37157	
37158	
37159	
37160	
37161	
37162	
37163	
37164	
37165	
37166	
37167	

0000-0002	0-2	USR jump
0003-0004	3-4	Float-Fixed vector
0005-0006	5-6	Fixed-Float vector
0007	7	Search character
0008	8	Scan-quotes flag
0009	9	TAB column save
000A	10	0=LOAD, 1=VERIFY
000B	11	Input buffer pointer/# subscript
000C	12	Default DIM flag
000D	13	Type: FF=string, 00=numeric
000E	14	Type: 80=integer, 00=floating point
000F	15	DATA scan/LIST quote/memory flag
0010	16	Subscript/FNx flag
0011	17	0=INPUT;$40=GET;$98=READ
0012	18	ATN sign/Comparison eval flag
0013	19	Current I/O prompt flag
0014-0015	20-21	Integer value
0016	22	Pointer: temporary string stack
0017-0018	23-24	Last temp string vector
0019-0021	25-33	Stack for temporary strings
0022-0025	34-37	Utility pointer area
0026-002A	38-42	Product area for multiplication
002B-002C	43-44	Pointer: Start-of-Basic
002D-002E	45-46	Pointer: Start-of-Variables
002F-0030	47-48	Pointer: Start-of-Arrays
0031-0032	49-50	Pointer: End-of-Arrays
0033-0034	51-52	Pointer: String-storage(moving down)
0035-0036	53-54	Utility string pointer
0037-0038	55-56	Pointer: Limit-of-memory
0039-003A	57-58	Current Basic line number
003B-003C	59-60	Previous Basic line number
003D-003E	61-62	Pointer: Basic statement for CONT
003F-0040	63-64	Current DATA line number
0041-0042	65-66	Current DATA address
0043-0044	67-68	Input vector
0045-0046	69-70	Current variable name
0047-0048	71-72	Current variable address
0049-004A	73-74	Variable pointer for FOR/NEXT
004B-004C	75-76	Y-save; op-save; Basic pointer save
004D	77	Comparison symbol accumulator
004E-0053	78-83	Misc work area, pointers, etc
0054-0056	84-86	Jump vector for functions
0057-0060	87-96	Misc numeric work area
0061	97	Accum#1: Exponent
0062-0065	98-101	Accum#1: Mantissa
0066	102	Accum#1: Sign
0067	103	Series evaluation constant pointer
0068	104	Accum#1 hi-order (overflow)
0069-006E	105-110	Accum#2: Exponent, etc.

0071-0072	113-114	Cassette buff len/Series pointer
0073-008A	115-138	CHRGET subroutine; get Basic char
007A-007B	122-123	Basic pointer (within subrtn)
008B-008F	139-143	RND seed value
0090	144	Status word ST
0091	145	Keyswitch PIA: STOP and RVS flags
0092	146	Timing constant for tape
0093	147	Load=0, Verify=1
0094	148	Serial output: deferred char flag
0095	149	Serial deferred character
0096	150	Tape EOT received
0097	151	Register save
0098	152	How many open files
0099	153	Input device, normally 0
009A	154	Output CMD device, normally 3
009B	155	Tape character parity
009C	156	Byte-received flag
009D	157	Direct=$80/RUN=0 output control
009E	158	Tp Pass 1 error log/char buffer
009F	159	Tp Pass 2 err log corrected
00A0-00A2	160-162	Jiffy Clock HML
00A3	163	Serial bit count/EOI flag
00A4	164	Cycle count
00A5	165	Countdown,tape write/bit count
00A6	166	Tape buffer pointer
00A7	167	Tp Wrt ldr count/Rd pass/inbit
00A8	168	Tp Wrt new byte/Rd error/inbit cnt
00A9	169	Wrt start bit/Rd bit err/stbit
00AA	170	Tp Scan;Cnt;Ld;End/byte assy
00AB	171	Wr lead length/Rd checksum/parity
00AC-00AD	172-173	Pointer: tape buff, scrolling
00AE-00AF	174-175	Tape end adds/End of program
00B0-00B1	176-177	Tape timing constants
00B2-00B3	178-179	Pntr: start of tape buffer
00B4	180	TP Wrt timer enabled; bit cnt
00B5	181	Tp EOT/RS232 next bit to send
00B6	182	Read character error/outbyte buf
00B7	183	# characters in file name
00B8	184	Current logical file
00B9	185	Current secndy address
00BA	186	Current device
00BB-00BC	187-188	Pointer to file name
00BD	189	Wr shift word/Rd input char
00BE	190	# blocks remaining to Wr/Rd
00BF	191	Serial word buffer
00C0	192	Tape motor interlock
00C1-00C2	193-194	I/O start addr
00C3-00C4	195-196	Kernel setup pointer
00C5	197	Last key pressed

00C7	199	Screen reverse flag
00C8	200	End-of-line for input pointer
00C9-00CA	201-202	Input cursor log (row, column)
00CB	203	Which key: 64 if no key
00CC	204	0=flash cursor
00CD	205	Cursor timing countdown
00CE	206	Character under cursor
00CF	207	Cursor in blink phase
00D0	208	Input from screen/from keyboard
00D1-00D2	209-210	Pointer to screen line
00D3	211	Position of cursor on above line
00D4	212	0=direct cursor, else programmed
00D5	213	Current screen line length
00D6	214	Row where cursor lives
00D7	215	Last inkey/checksum/buffer
00D8	216	# of INSERTs outstanding
00D9-00F0	217-240	Screen line link table
00F1	241	Dummy screen link
00F2	242	Screen row marker
00F3-00F4	243-244	Screen color pointer
00F5-00F6	245-246	Keyboard pointer
00F7-00F8	247-248	RS-232 Rcv pntr
00F9-00FA	249-250	RS-232 Tx pntr
00FF-010A	256-266	Floating to ASCII work area
0100-103E	256-318	Tape error log
0100-01FF	256-511	Processor stack area
0200-0258	512-600	Basic input buffer
0259-0262	601-610	Logical file table
0263-026C	611-620	Device # table
026D-0276	621-630	Sec Adds table
0277-0280	631-640	Keybd buffer
0281-0282	641-642	Start of Basic Memory
0283-0284	643-644	Top of Basic Memory
0285	645	Serial bus timeout flag
0286	646	Current color code
0287	647	Color under cursor
0288	648	Screen memory page
0289	649	Max size of keybd buffer
028A	650	Repeat all keys
028B	651	Repeat speed counter
028C	652	Repeat delay counter
028D	653	Keyboard Shift/Control flag
028E	654	Last shift pattern
028F-0290	655-656	Keyboard table setup pointer
0291	657	Keymode (Kattacanna)
0292	658	0=scroll enable
0293	659	VIC chip control
0294	660	VIC chip command
0295-0296	661-662	Bit timing
0297	663	RS-232 status

0298	664	RS-232 speed/code
0299	665	RS232 receive pointer
029A	666	RS232 input pointer
029B	667	RS232 transmit pointer
029C	668	RS232 output pointer
029D	669	IRQ save during tape I/O
029E	670	Error message link
029F-02A0	671-672	Basic warm start link
0300-0301	768-769	Crunch Basic tokens link
0302-0303	770-771	Print tokens link
0304-0305	772-773	Start new Basic code link
0306-0307	774-775	Get arithmetic element link
0308-0309	776-777	Hardware interrupt vector (EAHF)
030A-030B	778-779	Break interrupt vector (FED2)
030C	780	NMI interrupt vector (FEAD)
030D	781	OPEN vector (F40A)
030E	782	CLOSE vector (F34A)
030F	783	Set-input vector (F2C7)
0310	784	Set-output vector (F309)
0311	785	Restore I/O vector (F3F3)
0312-0313	786-787	INPUT vector (F20E)
0314-0315	788-789	Output vector (F27A)
0316-0317	790-791	Test-STOP vector (F770)
0318-0319	792-793	GET vector (F1F5)
031A-031B	794-795	Abort I/O vector (F3EF)
031C-031D	796-797	USER vector (FED2)
031E-031F	798-799	Set-timeout vector
0320-0321	800-801	Unused
0322-0323	802-803	Unused
0324-0325	804-805	Unused
0326-0327	806-807	Unused
0328-0329	808-809	Unused
032A-032B	810-811	Unused
032C-032D	812-813	Unused
032E-032F	814-815	Unused
0330-0331	816-817	LOAD link
0332-0333	818-819	SAVE link
033C-033F	828-1019	Cassette buffer
0400-0FFF	1024-4095	3K RAM expansion area
1000-1FFF	4096-8191	Normal Basic memory
2000-7FFF	8192-32767	Memory expansion area
8000-8FFF	32768-36863	Character bit maps
9000-900F	36864-36879	Video interface chip
9110-912F	37136-37151	VIA interface - NMI
9120-912F	37152-37167	VIA interface - IRQ
9400-95FF	37888-38399	Alternate Colour Wybble area.
9600-97FF	38400-38911	Main Colour Wybble area
A000-BFFF	40960-49151	Plug-in ROM area
C000-FFFF	49152-65535	ROM: Basic and Operating System
FFBA-FFF5	65418-65525	Jump Table, including:
		FFC6 - Set Input channel
		FFC9 - Set Output channel
		FFCC - Restore default I/O channels
		FFCF - INPUT
		FFD2 - PRINT
		FFE1 - Test Stop key
		FFE4 - GET
for more, see VIC Programmers Reference Guide, pg 184		

VIC-20 ROM Organization

c000	ROM control vectors	
c00c	Keyword action vectors	
c052	Function vectors	
c080	Operator vectors	
c09e	Keywords	
c19e	Error messages	
c328	Error message vectors	
c365	Miscellaneous messages	
c38a	Scan stack for FOR/GOSUB	
c3b8	Move memory	
c3fb	Check stack depth	
c408	Check memory space	
c435	'OUT OF MEMORY'	
c437	Error routine	
c469	Break entry	
c474	'READY.'	
c480	Ready for Basic	
c49c	Handle new line	
c533	Re-chain lines	
c560	Receive input line	
c579	Crunch tokens	
c613	Find Basic line	
c642	Perform [NEW]	
c65e	Perform [CLR]	
c68e	Back up text pointer	
c69c	Perform [LIST]	
c742	Perform [FOR]	
c76d	Execute statement	
c81d	Perform [RESTORE]	
c82c	Break	
c82f	Perform [STOP]	
c831	Perform [END]	
c857	Perform [CONT]	
c871	Perform [RUN]	
c883	Perform [GOSUB]	
c8a0	Perform [GOTO]	
c8d2	Perform [RETURN]	
c8f8	Perform [DATA]	
c906	Scan for next statement	
c938	Perform [IF]	
c93b	Perform [REM]	
c94b	Perform [ON]	
c96b	Get fixed point number	
c9a5	Perform [LET]	
c9d0	Perform [PRINT#]	
ca86	Perform [CMD]	
caa0	Perform [PRINT]	
cb1e	Print message from (y,a)	
cb3b	Print format character	
cb4d	Bad-input routines	
cb7b	Perform [GET]	
cba5	Perform [INPUT#]	
cbbf	Perform [INPUT]	
cbf9	Prompt & input	
cc06	Perform [READ]	
ccfc	Input error messages	
cd1e	Perform [NEXT]	
cd78	Type-match check	
cd9e	Evaluate expression	
ce48	Constant - PI	
ce51	Evaluate within brackets	
ce67	Check for ')'	
ce6f	Check for comma	
cf08	Syntax error	
cf14	Check range	
cf28	Search for variable	
cfa7	Set up FN reference	
cfe6	Perform [OR]	
cfe9	Perform [AND]	
d016	Compare	
d081	Perform [DIM]	
d08b	Locate variable	
d113	Check alphabetic	
d11d	Create variable	
d194	Array pointer subroutine	
d1a5	Value 32768	
d1b2	Float-fixed conversion	
d1d1	Set up array	
d245	'BAD SUBSCRIPT'	
d248	'ILLEGAL QUANTITY'	
d14c	Compute array size	
d37d	Perform [FRE]	
d391	Fixed-float conversion	
d39e	Perform [POS]	
d3a6	Check direct	
d3b3	Perform [DEF]	
d3e1	Check FN syntax	
d3f4	Perform [FN]	
d465	Perform [STR$]	
d475	Calculate string vector	
d487	Set up string	
d4f4	Make room for string	
d526	Garbage collection	
d5bd	Check salvageability	
d606	Collect string	
d63d	Concatenate	
d67a	Build string to memory	
d6a3	Discard unwanted string	
d6db	Clean descriptor stack	
d6ec	Perform [CHR$]	
d700	Perform [LEFT$]	
d72c	Perform [RIGHT$]	
d737	Perform [MID$]	
d761	Pull string parameters	
d77c	Perform [LEN]	
d782	Exit string-mode	
d78b	Perform [ASC]	
d79b	Input byte parameter	
d7ad	Perform [VAL]	
d7eb	Get params for poke/wait	
d7f7	Float-fixed	
d80d	Perform [PEEK]	
d824	Perform [POKE]	
d82d	Perform [WAIT]	
d849	Add 0.5	
d850	Subtract-from	
d853	Perform [SUBTRACT]	
d86a	Perform [ADD]	
d947	Complement fac#1	
d97e	'OVERFLOW'	
d983	Multiply by zero byte	
d9ea	Perform [LOG]	
da2b	Perform [MULTIPLY]	
da59	Multiply-a-bit	
da8c	Memory to FAC#2	
dab7	Adjust FAC#1/#2	
dad4	Underflow/overflow	
dae2	Multiply by 10	
daf9	*10 in floating pt	
dafe	Divide by 10	
db12	Perform [DIVIDE]	
dba2	Memory to fac#1	
dbe7	FAC#1 to memory	
dbfc	FAC#2 to fac#1	
dc0c	FAC#1 to FAC#2	
dc1b	Round FAC#1	
dc2b	Get sign	
dc39	Perform [SGN]	
dc58	Perform [ABS]	
dc5b	Compare FAC#1 to mem	
dc9b	Float-fixed	
dccc	Perform [INT]	
dcf3	String to fac	
dd7e	Get ascii digit	
dddd	Float to ascii	
df16	Decimal constants	
df3a	TI constants	
df71	Perform [SQR]	
df7b	Perform [POWER]	
dfb4	Perform [NEGATIVE]	
dfed	Perform [EXP]	
e040	Series evaluate 1	
e056	Series evaluate 2	
e094	Perform [RND]	
e0f6	?? Breakpoints ??	
e127	Perform [SYS]	
e153	Perform [SAVE]	
e162	Perform [VERIFY]	
e165	Perform [LOAD]	
e1bb	Perform [OPEN]	
e1c4	Perform [CLOSE]	
e1d1	Params for load/save	
e203	Check default params	
e20b	Check for comma	
e216	Params for open/close	
e261	Perform [COS]	
e268	Perform [SIN]	
e2b1	Perform [TAN]	
e30b	Perform [ATN]	
e378	Initialize	
e387	CHRGET for zero page	
e3a4	Initialize Basic	
e429	Power-up message	
e44f	Vectors for $300	
e45b	Initialize vectors	
e467	Warm restart	
e476	Program patch area	
e4a0	Serial output '1'	
e4a9	Serial output '0'	
e4b2	Get serial input & clock	
e4bc	Program patch area	
e500	Set screen limits	
e50a	Track cursor location	
e518	Initalise I/O	
e54c	Normalise screen	
e55f	Clear screen	
e581	Home cursor	
e587	Set screen pointers	
e5bb	Set I/O defaults	
e5e3	Set vic chip defaults	
e5ef	Input from keyboard	
e64f	Input from screen	
e6b8	Quote mark test	
e6c5	Set up screen print	
e6ea	Advance cursor	
e715	Retreat cursor	
e72d	Back into previous line	
e742	Output to screen	
e8c3	Go to next line	
e8d8	Do 'RETURN'	
e8e8	Check line decrement	
e8fa	Check line increment	
e912	Set colour code	
e921	Colour code table	
e929	Code conversion	
e975	Scroll screen	
e9ee	Open space on screen	
ea56	Move screen line	
ea6e	Synch colour transfer	
ea7e	Set start-of-line	
ea8d	Clear screen line	
eaa1	Print to screen	
eaea	Store on screen	
eab2	Synch colour to char	
eabf	Interrupt (IRQ)	
eb1e	Check keyboard	
ec00	Set text mode	
ec46	Keyboard vectors	
ec5e	Keyboard maps	
ed21	Graphics/text control	
ed30	Set graphics mode	
ed5b	Wrap up screen line	
ed6a	Shifted key matrix	
ede3	Control key matrix	
ede4	Vic chip defaults	
edfd	Screen line adds low	
ee14	Send 'talk'	
ee17	Send 'listen'	
ee1e	Send control char	
ee49	Send to serial bus	
eeb7	Timeout on serial	
eec0	Send listen SA	
eec5	Clear ATN	
eece	Send talk SA	
eee4	Send serial deferred	
eef6	Send 'untalk'	
ef04	Send 'unlisten'	
ef19	Receive from serial bus	
ef84	Clock line on	
ef8d	Clock line off	
ef96	Delay 1 ms	
efa3	RS232 send (NMI)	
efee	New RS232 byte send	
f016	Error or quit	
f027	Compute bit count	
f036	RS232 receive (NMI)	
f05b	Setup to receive	
f09d	Receive parity error	
f0a2	Receive overrun error	
f0a5	Receive break error	
f0a8	Receive frame error	
f0b9	Bad device	
f0bc	File to RS232	
f0ed	Send to RS232 buffer	
f116	Input from RS232 buffer	
f14f	Get from RS232 buffer	
f160	Check serial bus idle	
f174	Messages	
f1e2	Print if direct	
f1f5	Get..	
f205	..from RS232	
f20e	Input	
f250	Get.. tape/serial/RS232	
f27a	Output..	
f290	..to tape	
f2e7	Set input device	
f309	Set output device	
f34a	Close	
f3cf	Find file	
f3df	Set file values	
f3ef	Abort all files	
f3f3	Restore default I/O	
f40a	Do file opening	
f495	Send SA	
f4e7	Open RS232	
f542	Load program	
f647	'SEARCHING'	
f659	Print file name	
f66a	'LOADING/VERIFYING'	
f675	Save program	
f72a	'SAVING'	
f734	Bump clock	
f760	Get time	
f767	Set time	
f770	Action stop key	
f77e	File Error Messages	
f7af	Find any tape header	
f7e7	Write tape header	
f84d	Get buffer address	
f854	Set buffer start, end pntrs	
f867	Find specific header	
f88a	Bump tape pointer	
f894	'PRESS PLAY .. '	
f8ab	Check cassette status	
f8b7	'PRESS RECORD ..'	
f8c0	Initiate tape read	
f8e3	Initiate tape write	
f8f4	Common tape read/write	
f94b	Check tape stop	
f95d	Set timing	
f98e	Read bits (IRQ)	
faad	Store characters	
fbd2	Reset pointer	
fbdb	New tape character setup	
fbea	Toggle tape	
fc06	Data write	
fc0b	Tape write (IRQ)	
fc95	Leader write (IRQ)	
fccf	Restore vectors	
fcf6	Set vector	
fd08	Kill motor	
fd11	Check read/write pntr	
fd1b	Bump read/write pntr	
fd22	Powerup entry	
fd3f	Check r-w-vrfy	
fd52	Set kernal2	
fd8d	Initialise system consts	
fdf1	IRQ vectors	
fdf9	Initialise I/O regs	
fe49	Save data name	
fe50	Save file details	
fe57	Get status	
fe66	Flag ST	
fe6f	Set timeout	
fe73	Read/set top of memory	
fe82	Read/set bottom of mem	
fe91	Test memory location	
fea9	NMI interrupt entry	
fed2	RESET/STOP warm start	
fede	NMI RS232 sequences	
fe56	Restore & exit	
ff5c	RS232 timing table	
ff72	Main IRQ entry	
ff8a	Jumbo jump table	
fffa	Hardware vectors	